Microlinc, LLC v. Intel Corporatioh et al Doc. 1 Att. 1
Case 2:07-cv-00488-TJW Document 1 Filed 11/07/2007 Page 1 of 28

US006009488A

United States Patent [19]
Kavipurapu

[11] **Patent Number: 6,009,488**
[45] **Date of Patent: Dec. 28, 1999**

[54] **COMPUTER HAVING PACKET-BASED INTERCONNECT CHANNEL**

[75] Inventor: **Gautam Kavipurapu**, Dallas, Tex

[73] Assignee: **Microlinc, LLC**, Tex.

[21] Appl No: **08/965,760**

[22] Filed: **Nov. 7, 1997**

[51] **Int. Cl.**[6] **G06F 13/00**
[52] U.S. Cl. **710/105**; 710/129; 710/240; 710/29; 710/30; 710/36; 710/3
[58] **Field of Search** 710/3–7, 29–39, 710/105–127, 129–130, 240

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,777,595	10/1988	Strecker et al	395/200.66
5,191,657	3/1993	Ludwig et al	395/325
5,390,308	2/1995	Ware et al.	395/400
5,499,385	3/1996	Farmwald et al	395/823
5,555,381	9/1996	Ludwig et al	395/281
5,598,580	1/1997	Detschel et al.	395/855
5,606,717	2/1997	Farmwald et al	395/856
5,646,941	7/1997	Nishimura et al	370/389
5,809,263	9/1998	Farmwald et al	395/309
5,872,996	2/1999	Barth et al.	395/853

Primary Examiner—Christopher B. Shin
Attorney, Agent, or Firm—Lockwood, Alex, FitzGibbon & Cummings

[57] **ABSTRACT**

A physically non-distributed microprocessor-based computer includes a microprocessor, and a random access memory device, a mass storage device, and an input-output port device, all operable from the microprocessor and including an interface for receiving and transmitting data in packet form. A novel packet-based data channel extends between the microprocessor and the interfaces of the devices to provide communication between the microprocessor and the devices. By varying the size of the packets in accordance with actual data transmission requirements improved computer performance is achieved.

13 Claims, 21 Drawing Sheets

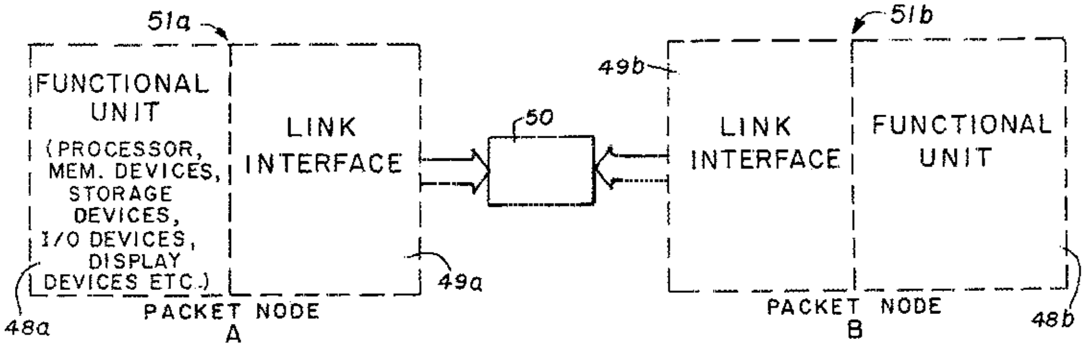

51a
FUNCTIONAL UNIT
(PROCESSOR, MEM. DEVICES, STORAGE DEVICES, I/O DEVICES, DISPLAY DEVICES ETC.)
LINK INTERFACE
49a
48a
PACKET NODE A
50
51b
49b
LINK INTERFACE
FUNCTIONAL UNIT
PACKET NODE B
48b

Exhibit A

Dockets.Justia.com

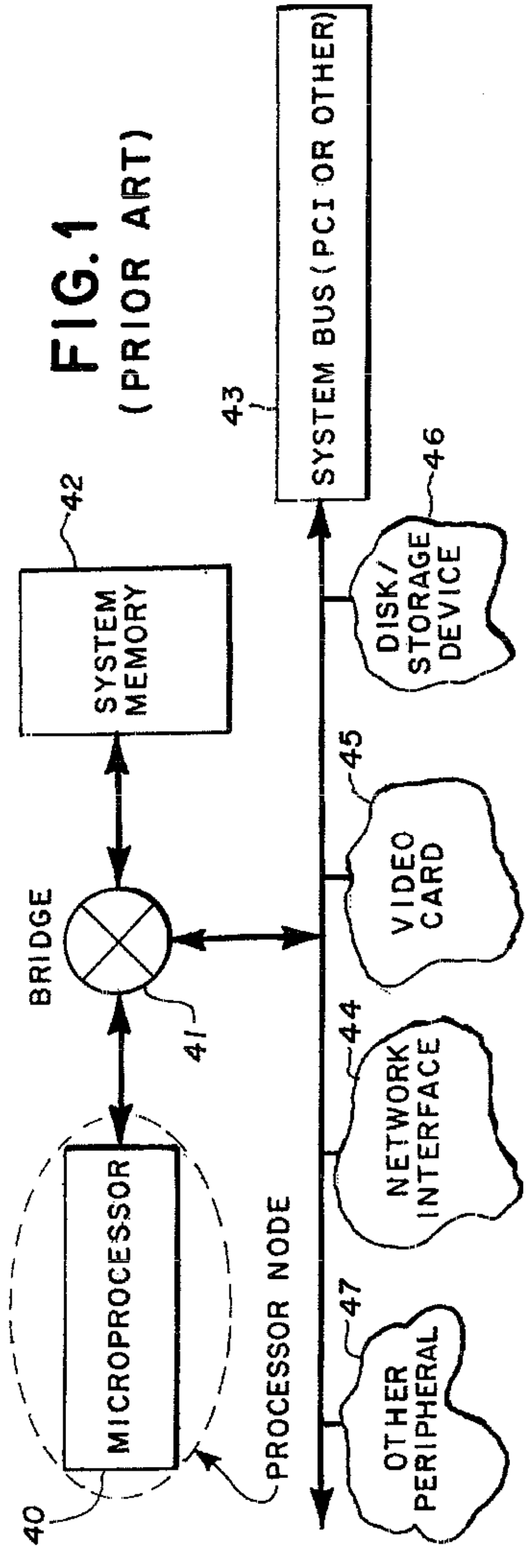

SYSTEM BUS (PCI OR OTHER)
43
SYSTEM MEMORY
42
BRIDGE
41
MICROPROCESSOR
40
PROCESSOR NODE
DISK/ STORAGE DEVICE
46
VIDEO CARD
45
NETWORK INTERFACE
44
OTHER PERIPHERAL
47

FIG. 1 (PRIOR ART)

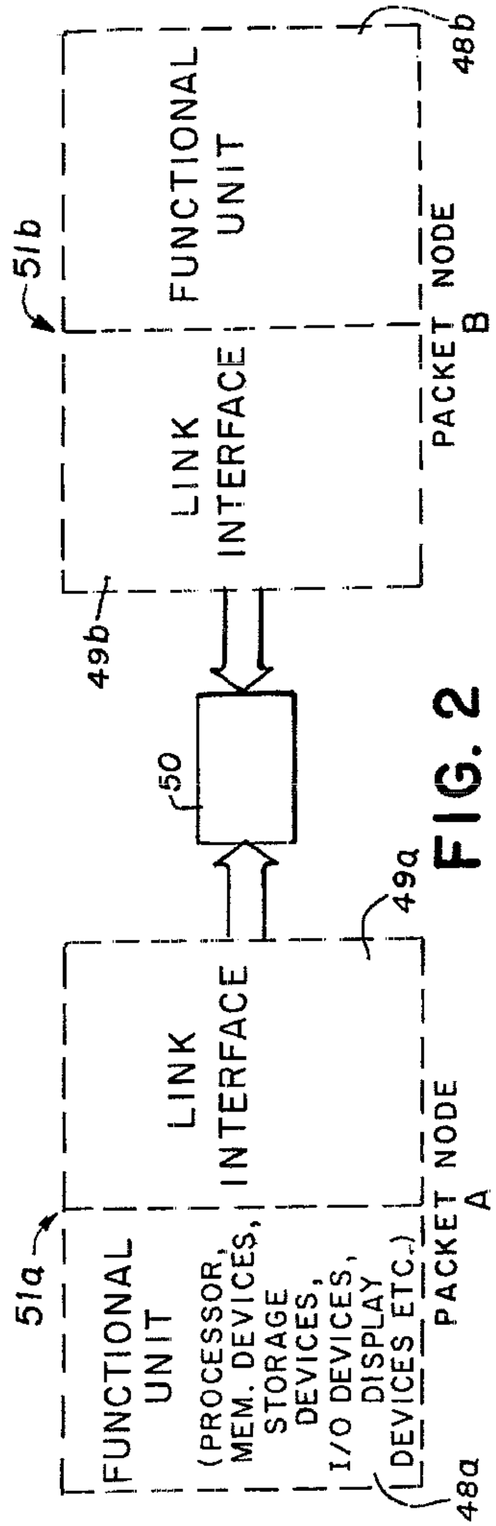

FUNCTIONAL UNIT
LINK INTERFACE
PACKET NODE B
48b
51b
49b
50
LINK INTERFACE
FUNCTIONAL UNIT (PROCESSOR, MEM. DEVICES, STORAGE DEVICES, I/O DEVICES, DISPLAY DEVICES ETC.)
PACKET NODE A
49a
51a
48a

FIG. 2

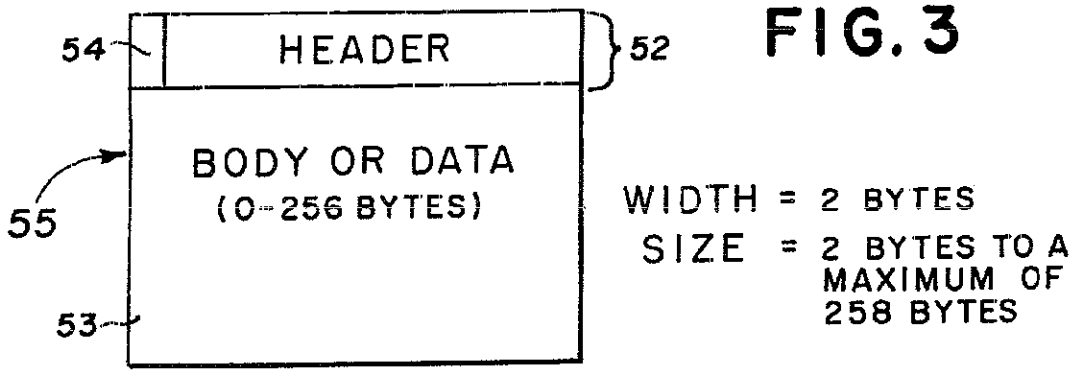

54
HEADER
52
BODY OR DATA
(0-256 BYTES)
55
53
WIDTH = 2 BYTES
SIZE = 2 BYTES TO A MAXIMUM OF 258 BYTES

FIG. 3

FIG. 4

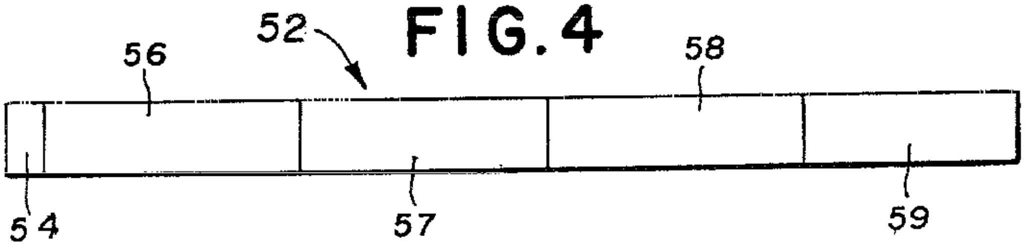
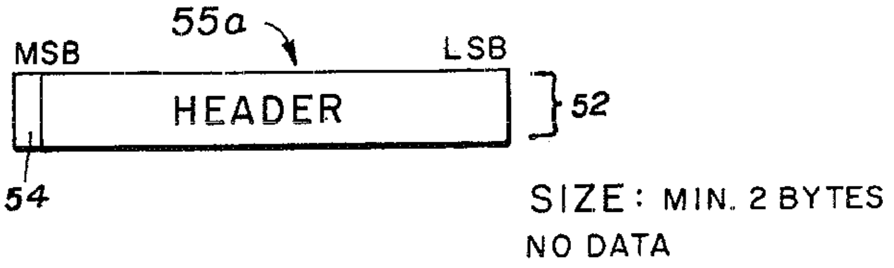

52
56
58
54
57
59

FIG. 5

IDLE PACKET


55a
MSB
LSB
HEADER
52
54
SIZE: MIN. 2 BYTES
NO DATA

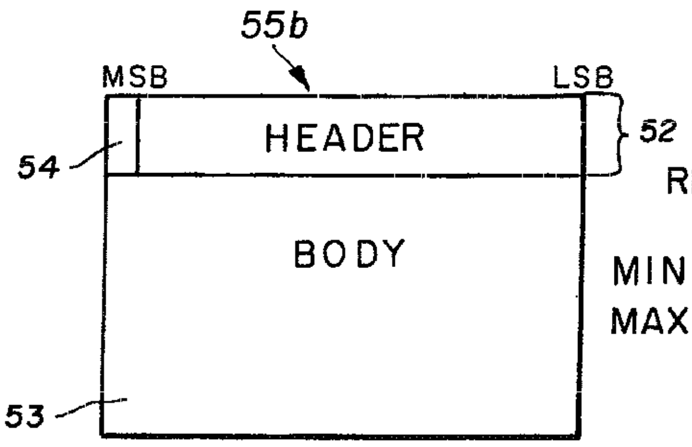
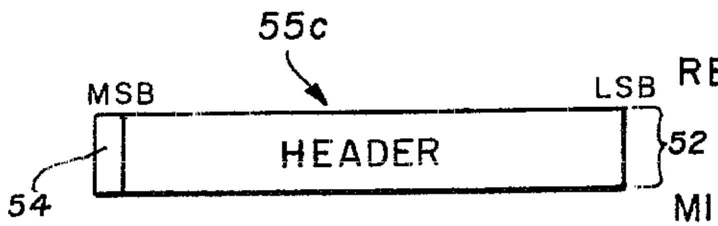

55b
MSB
LSB
HEADER
BODY
54
53
52

FIG. 6
REQUEST PACKET

MIN. SIZE: 4 BYTES
MAX. SIZE: 258 BYTES

FIG. 7
REQUEST ECHO PACKET


55c
MSB
LSB
HEADER
54
52

MIN. SIZE: 2 BYTES
NO DATA OR BODY

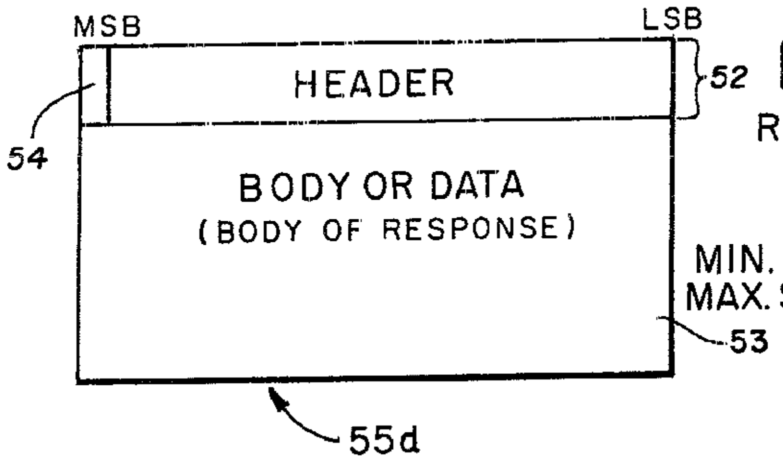

MSB
LSB
HEADER
BODY OR DATA
(BODY OF RESPONSE)
54
52
53
55d

FIG. 8
RESPONSE PACKET

MIN. SIZE: 4 BYTES
MAX. SIZE: 258 BYTES

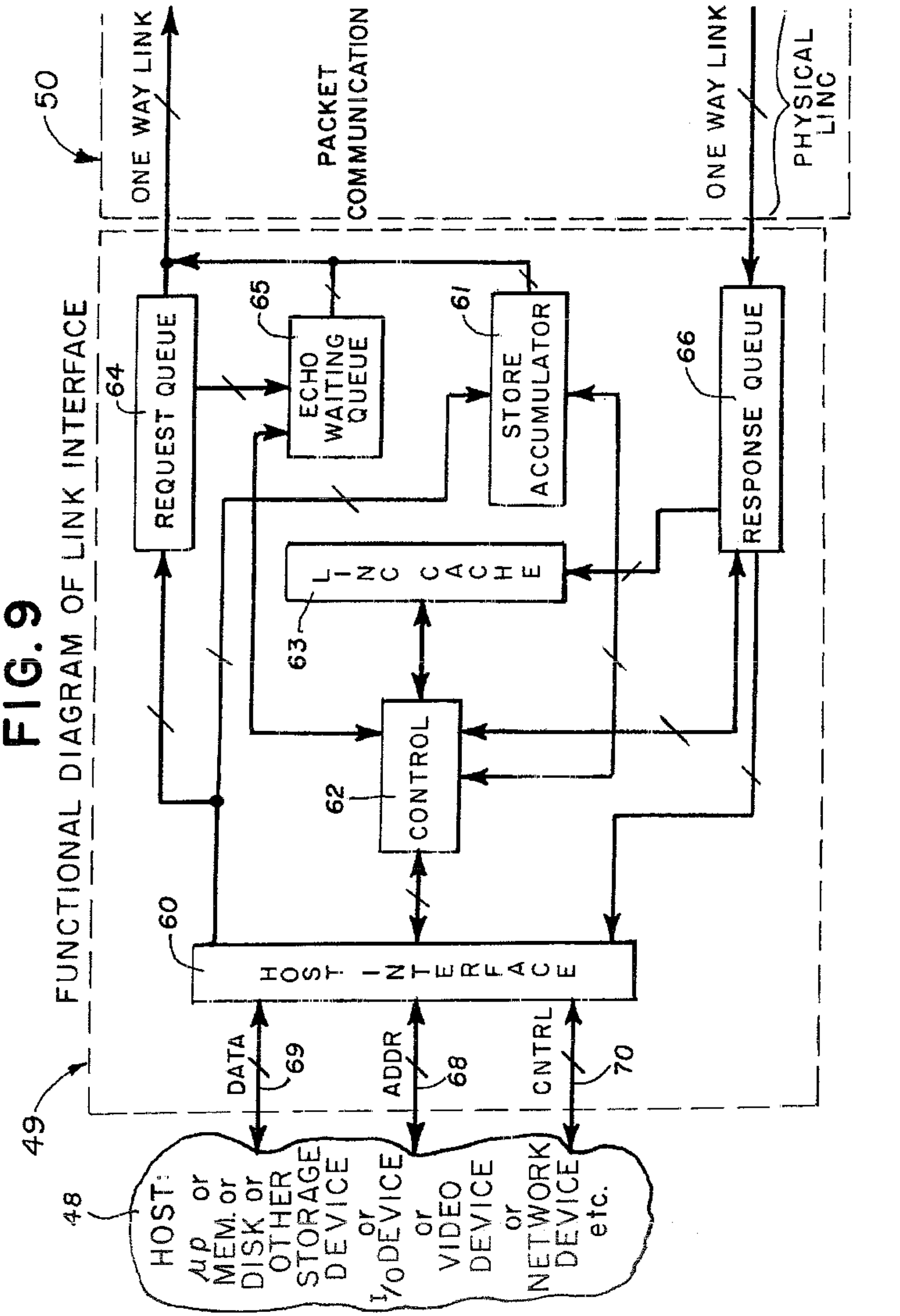

50
ONE WAY LINK
PACKET COMMUNICATION
ONE WAY LINK
PHYSICAL LINC
REQUEST QUEUE
64
ECHO WAITING QUEUE
65
STORE ACCUMULATOR
61
RESPONSE QUEUE
66
LINC CACHE
63
CONTROL
62
HOST INTERFACE
60
DATA
69
ADDR
68
CNTRL
70
49
48
HOST: μp or MEM. or DISK or OTHER STORAGE DEVICE or I/O DEVICE or VIDEO DEVICE or NETWORK DEVICE etc.

FIG. 9

FUNCTIONAL DIAGRAM OF LINK INTERFACE

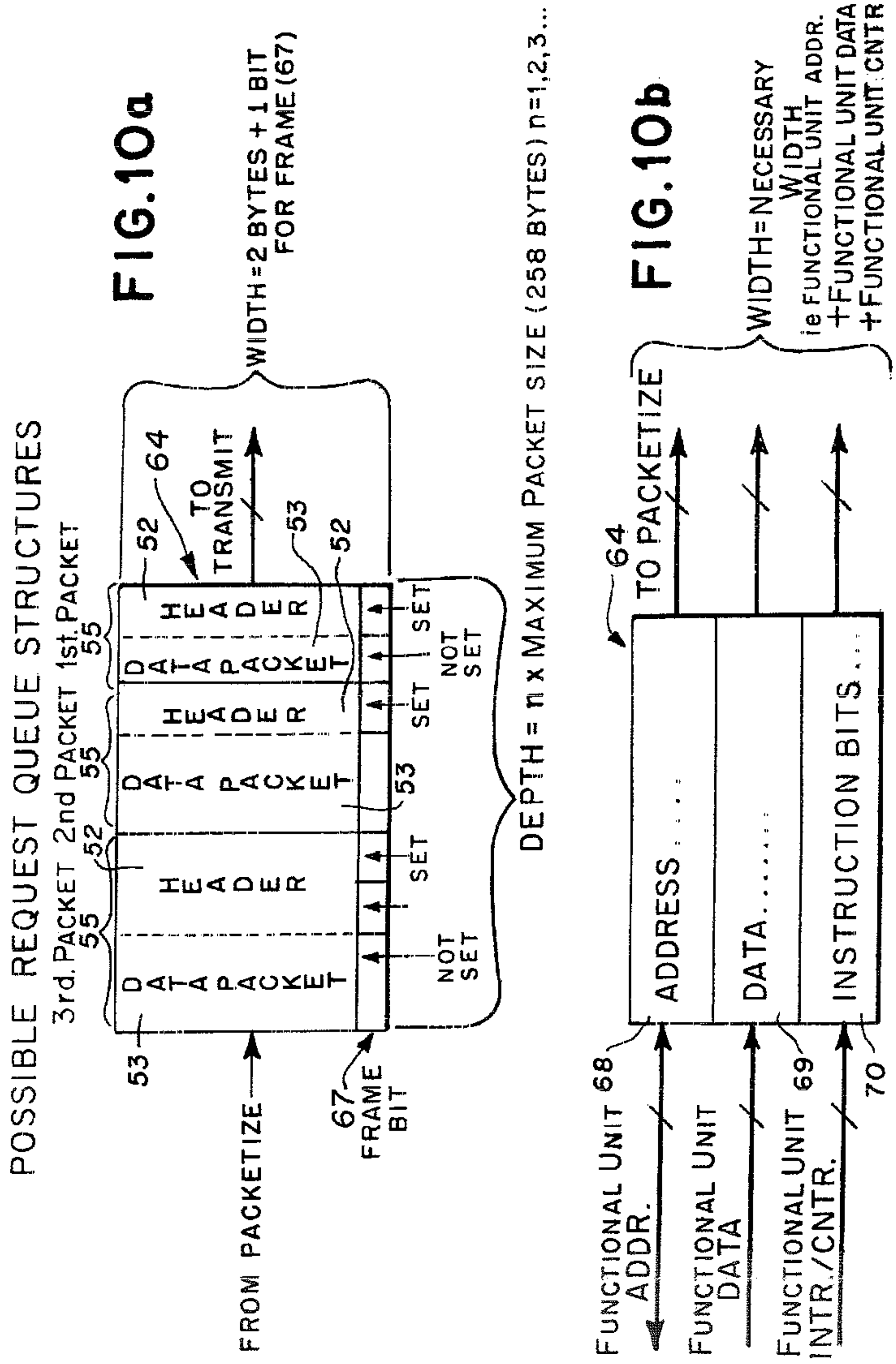

POSSIBLE REQUEST QUEUE STRUCTURES
3rd. PACKET
2nd PACKET
1st. PACKET
55
52
53
64
TO TRANSMIT
WIDTH = 2 BYTES + 1 BIT FOR FRAME (67)
HEADER
DATA PACKET
SET
NOT SET
FROM PACKETIZE
67
FRAME BIT
DEPTH = n x MAXIMUM PACKET SIZE (258 BYTES) n=1,2,3...
TO PACKETIZE
WIDTH=NECESSARY WIDTH
ie FUNCTIONAL UNIT ADDR.
+FUNCTIONAL UNIT DATA
+FUNCTIONAL UNIT CNTR
ADDRESS....
DATA.......
INSTRUCTION BITS...
68
69
70
FUNCTIONAL UNIT ADDR.
FUNCTIONAL UNIT DATA
FUNCTIONAL UNIT INTR./CNTR.

FIG. 10a

FIG. 10b

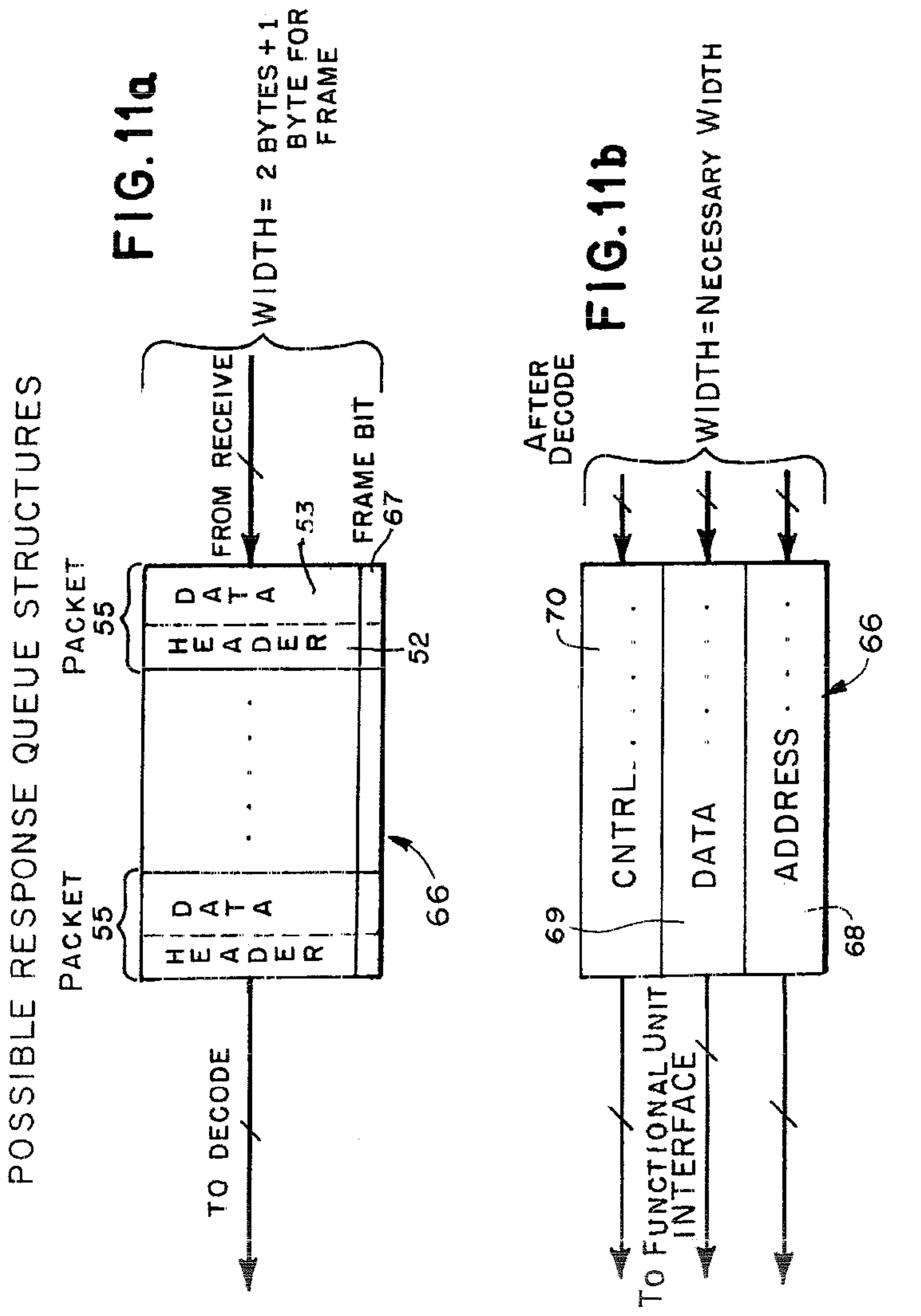
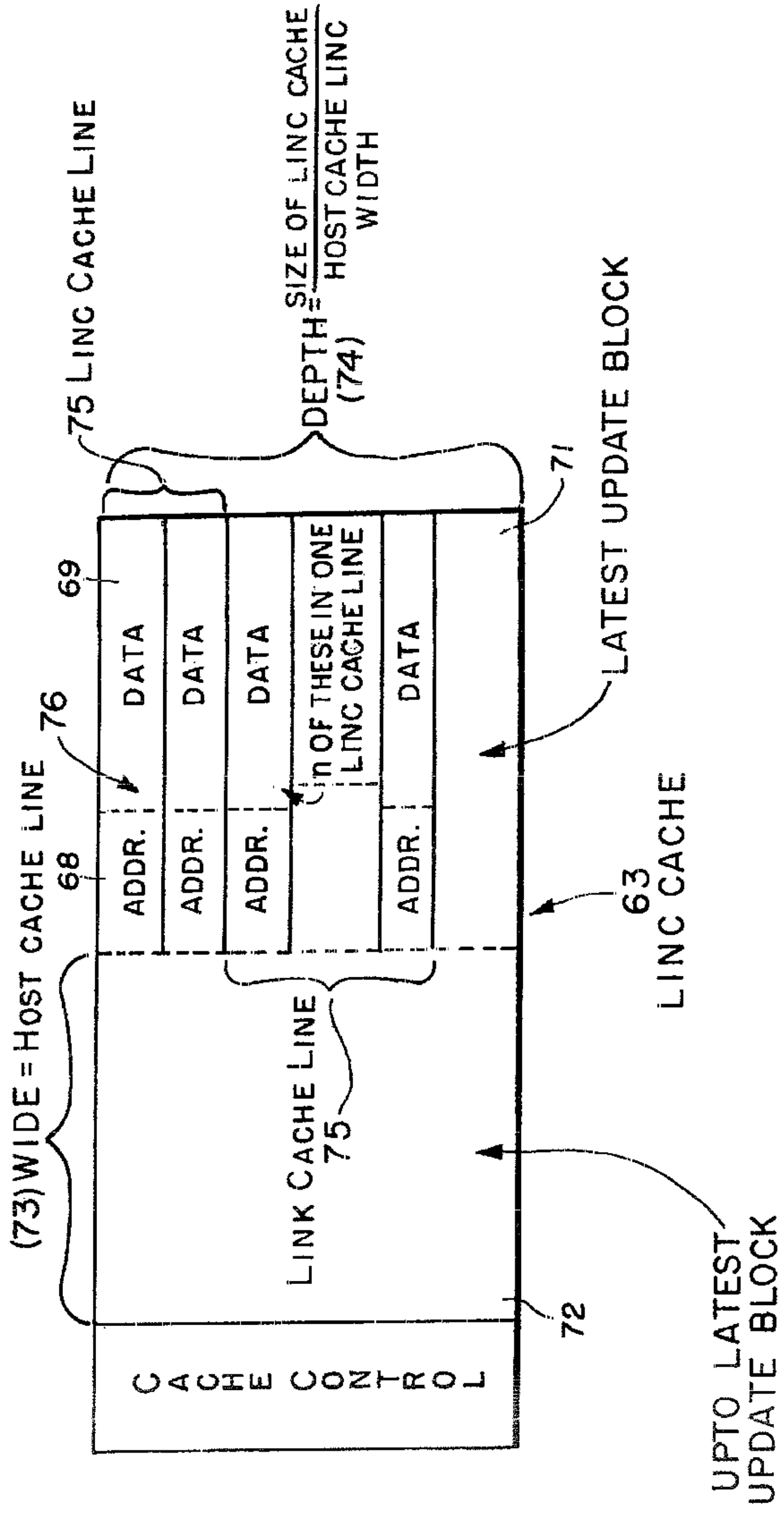


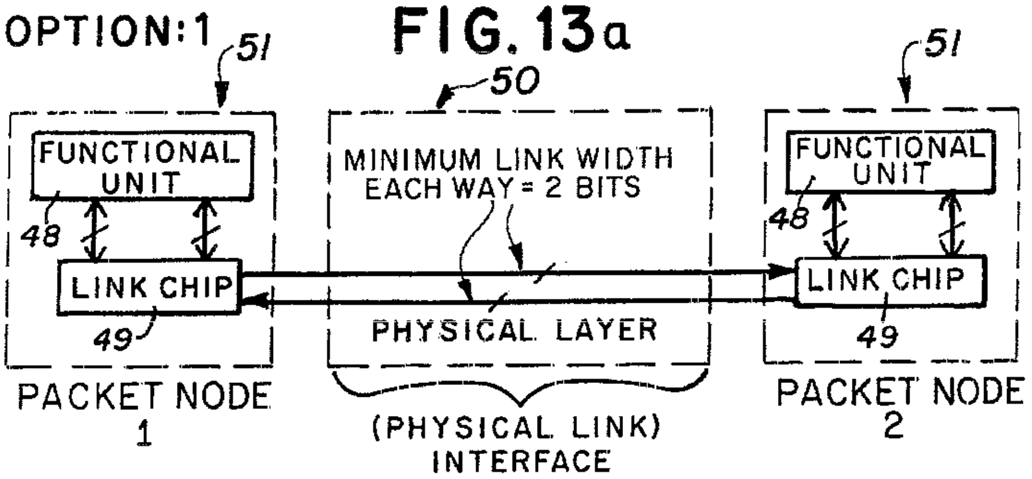
FIG. 11a
POSSIBLE RESPONSE QUEUE STRUCTURES
WIDTH = 2 BYTES + 1 BYTE FOR FRAME
FROM RECEIVE
FRAME BIT
67
53
PACKET 55
HEADER
DATA
52
66
TO DECODE
FIG. 11b
WIDTH = NECESSARY WIDTH
AFTER DECODE
70
CNTRL
DATA
ADDRESS
69
68
66
TO FUNCTIONAL UNIT INTERFACE

FIG.12


75 LINC CACHE LINE
DEPTH = SIZE OF LINC CACHE / HOST CACHE LINC WIDTH (74)
71
LATEST UPDATE BLOCK
69
DATA
DATA
DATA
n OF THESE IN ONE LINC CACHE LINE
DATA
76
ADDR.
ADDR.
ADDR.
ADDR.
68
(73) WIDE = HOST CACHE LINE
LINK CACHE LINE 75
63 LINC CACHE
72
UPTO LATEST UPDATE BLOCK
CACHE CONTROL

OPTION:1

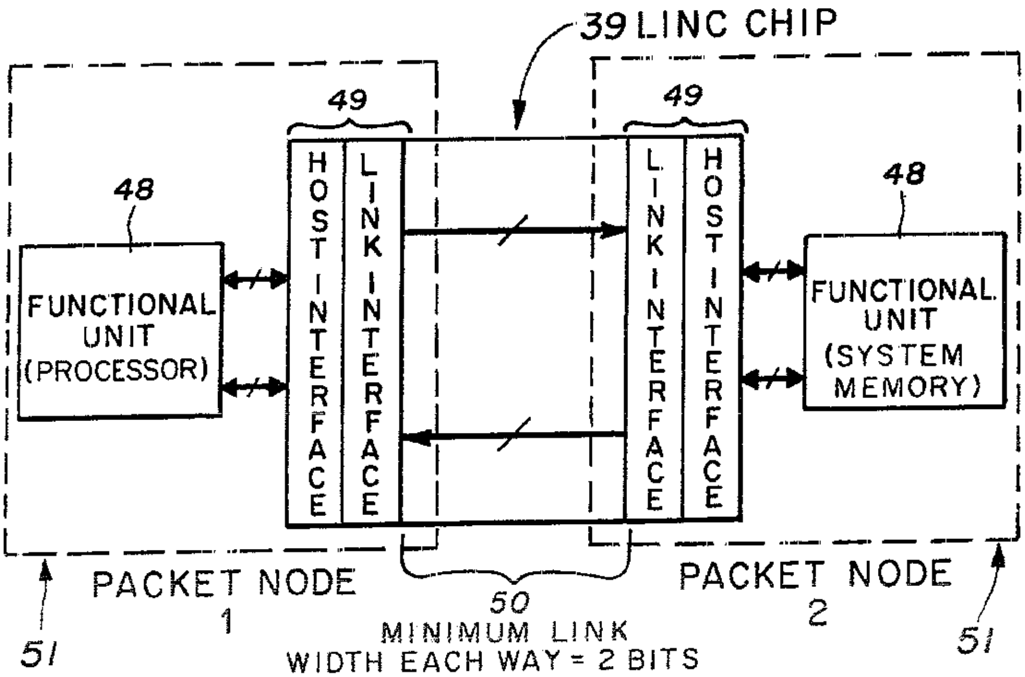
FIG. 13a

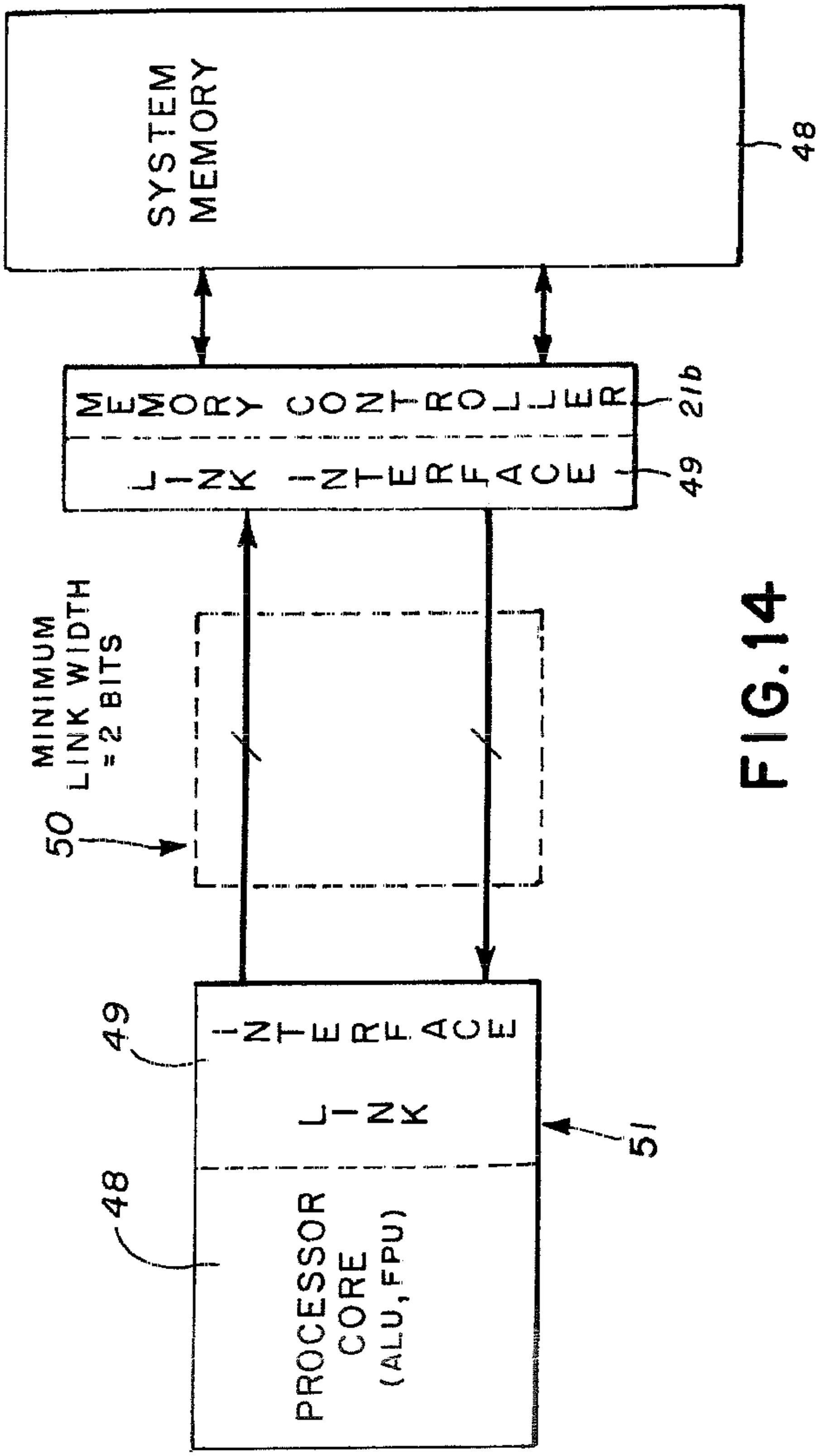

51
50
51
FUNCTIONAL UNIT
MINIMUM LINK WIDTH EACH WAY = 2 BITS
FUNCTIONAL UNIT
48
48
LINK CHIP
LINK CHIP
49
PHYSICAL LAYER
49
PACKET NODE 1
(PHYSICAL LINK) INTERFACE
PACKET NODE 2

A SAMPLE TWO NODE LINK WHERE NODE 1 FUNCTIONAL UNIT IS THE PROCESSOR AND NODE 2 FUNCTIONAL UNIT IS THE MEMORY.

OPTION:2

FIG. 13b


39 LINC CHIP
49
49
48
48
HOST INTERFACE
LINK INTERFACE
LINK INTERFACE
HOST INTERFACE
FUNCTIONAL UNIT (PROCESSOR)
FUNCTIONAL UNIT (SYSTEM MEMORY)
PACKET NODE 1
PACKET NODE 2
51
51
50
MINIMUM LINK WIDTH EACH WAY = 2 BITS



SYSTEM MEMORY
48
MEMORY CONTROLLER
21b
LINK INTERFACE
49
MINIMUM LINK WIDTH =2 BITS
50
INTERFACE
49
LINK
51
48
PROCESSOR CORE (ALU, FPU)

FIG.14

FIG.15

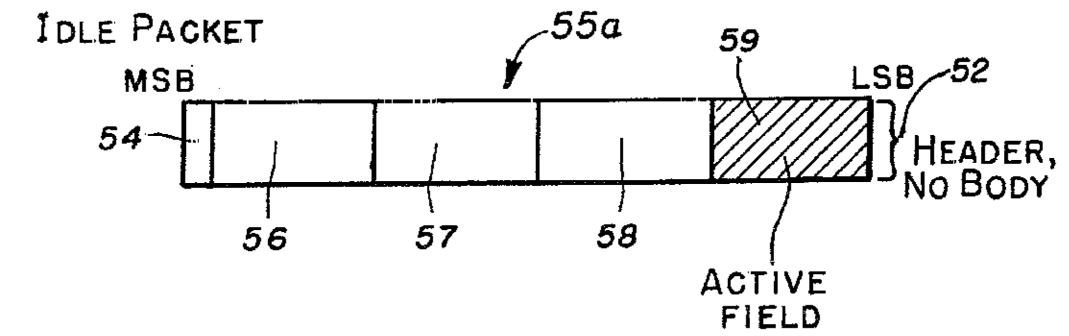

IDLE PACKET
55a
59
MSB
LSB
52
54
HEADER,
NO BODY
56
57
58
ACTIVE
FIELD

FIG.16

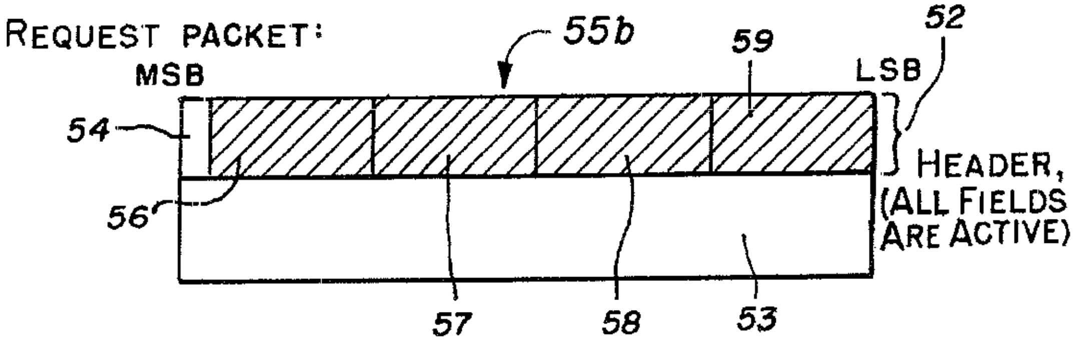

REQUEST PACKET:
55b
59
52
MSB
LSB
54
HEADER,
(ALL FIELDS
ARE ACTIVE)
56
57
58
53

FIG.17

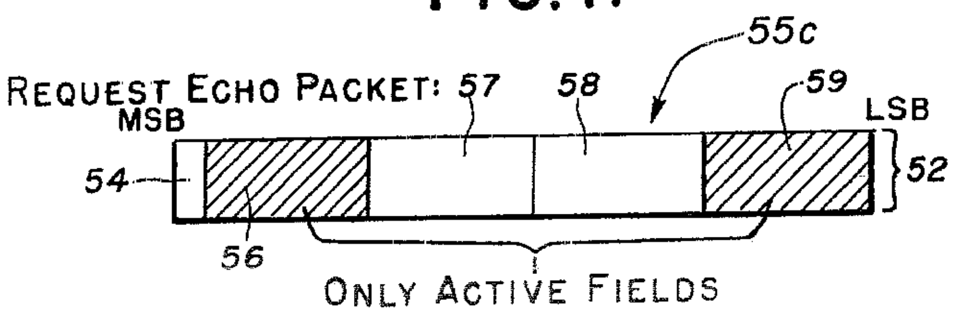

55c
REQUEST ECHO PACKET:
57
58
59
LSB
MSB
54
52
56
ONLY ACTIVE FIELDS

FIG.18

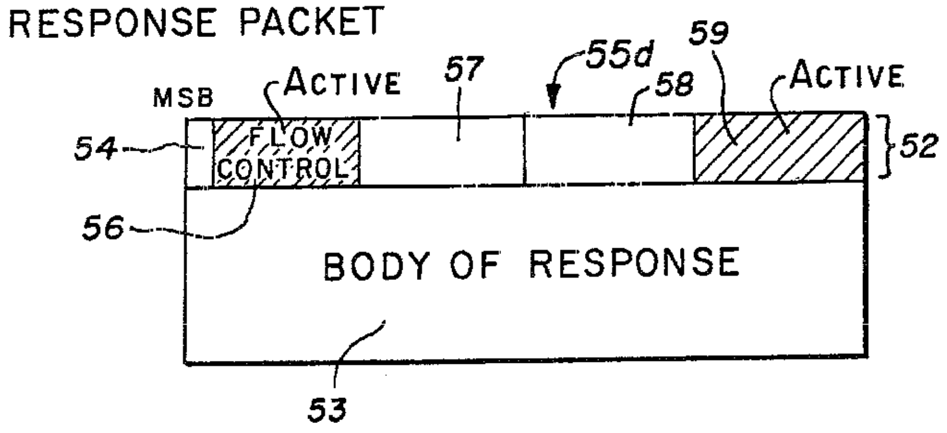
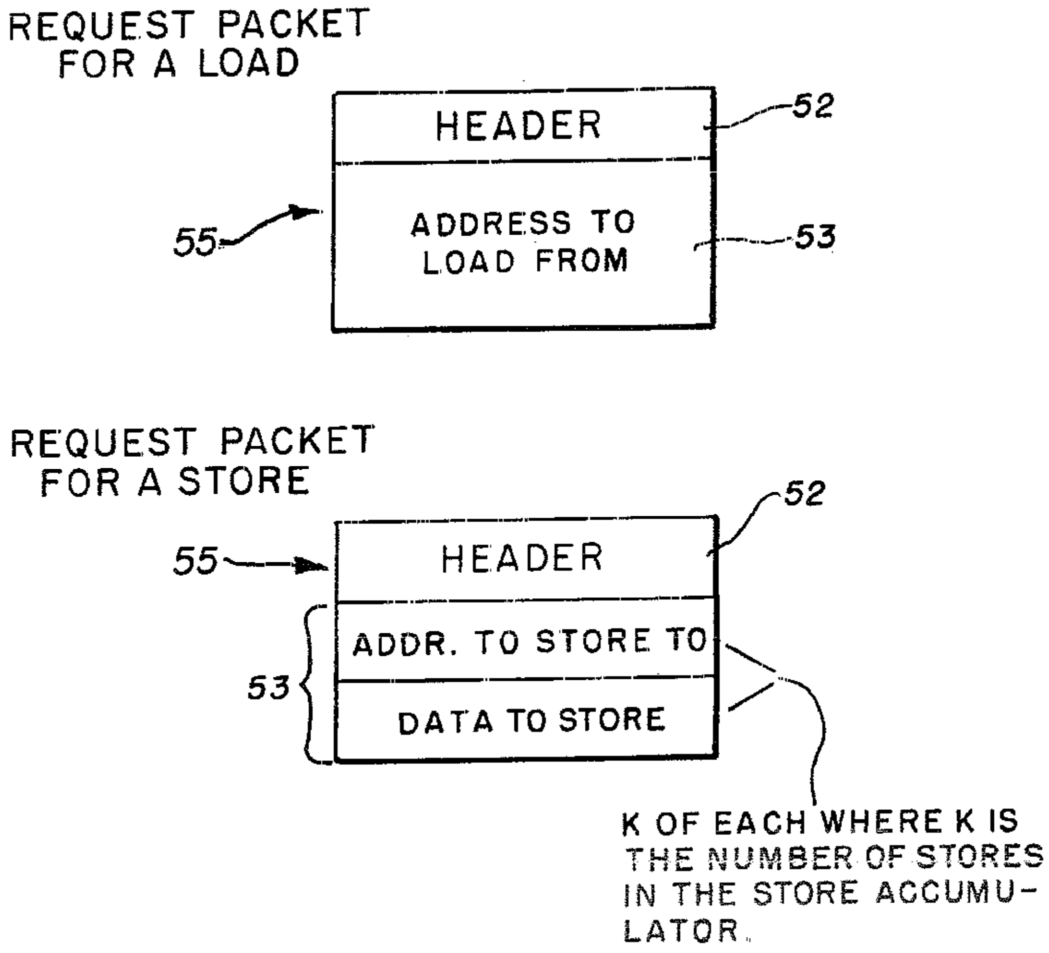
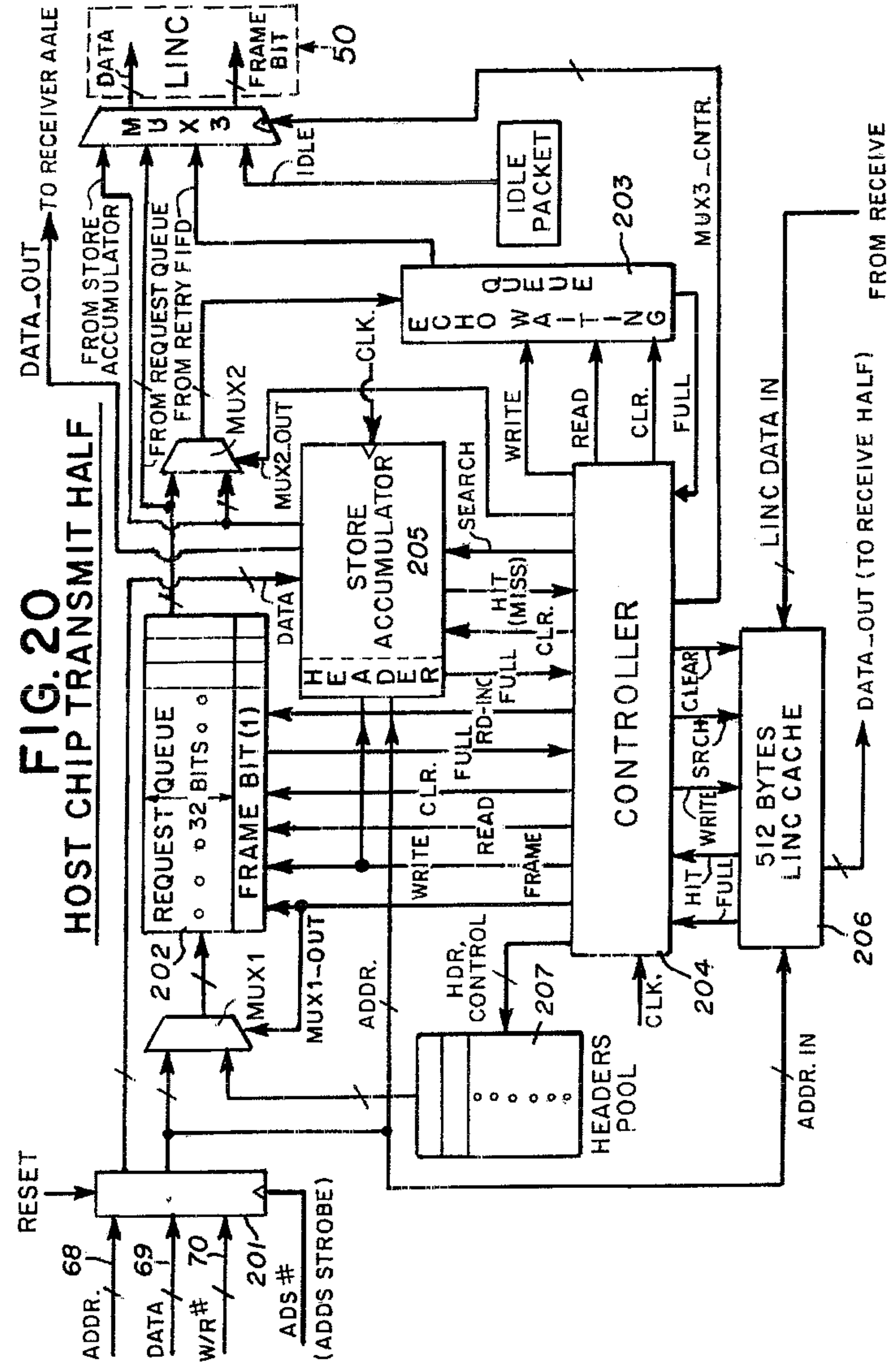

RESPONSE PACKET
MSB
ACTIVE
57
55d
59
58
ACTIVE
54
FLOW CONTROL
52
56
BODY OF RESPONSE
53

FIG.19


REQUEST PACKET FOR A LOAD
HEADER
52
55
ADDRESS TO LOAD FROM
53
REQUEST PACKET FOR A STORE
52
55
HEADER
ADDR. TO STORE TO
53
DATA TO STORE
K OF EACH WHERE K IS THE NUMBER OF STORES IN THE STORE ACCUMULATOR.



DATA_OUT
TO RECEIVER AALE
DATA
LINC
FRAME BIT
50
MUX 3
IDLE
IDLE PACKET
FROM STORE ACCUMULATOR
FROM REQUEST QUEUE
FROM RETRY FIFO
MUX3_CNTR.
203
ECHO WAITING QUEUE
CLK.
MUX2
MUX2_OUT
STORE ACCUMULATOR 205
HEADER
SEARCH
WRITE
READ
CLR.
FULL
FROM RECEIVE
LINC DATA IN
HIT (MISS)
CLR.
DATA
REQUEST QUEUE 32 BITS
FRAME BIT (1)
FULL
RD-INC
FULL
CLR.
READ
WRITE
FRAME
CONTROLLER
CLEAR
SRCH
WRITE
HIT
FULL
512 BYTES LINC CACHE
DATA_OUT (TO RECEIVE HALF)
206
202
MUX1
MUX1-OUT
ADDR.
HDR. CONTROL
207
CLK.
204
HEADERS POOL
ADDR. IN
RESET
68
69
70
201
ADDR.
DATA
W/R#
ADS#
(ADDS STROBE)

FIG. 20
HOST CHIP TRANSMIT HALF

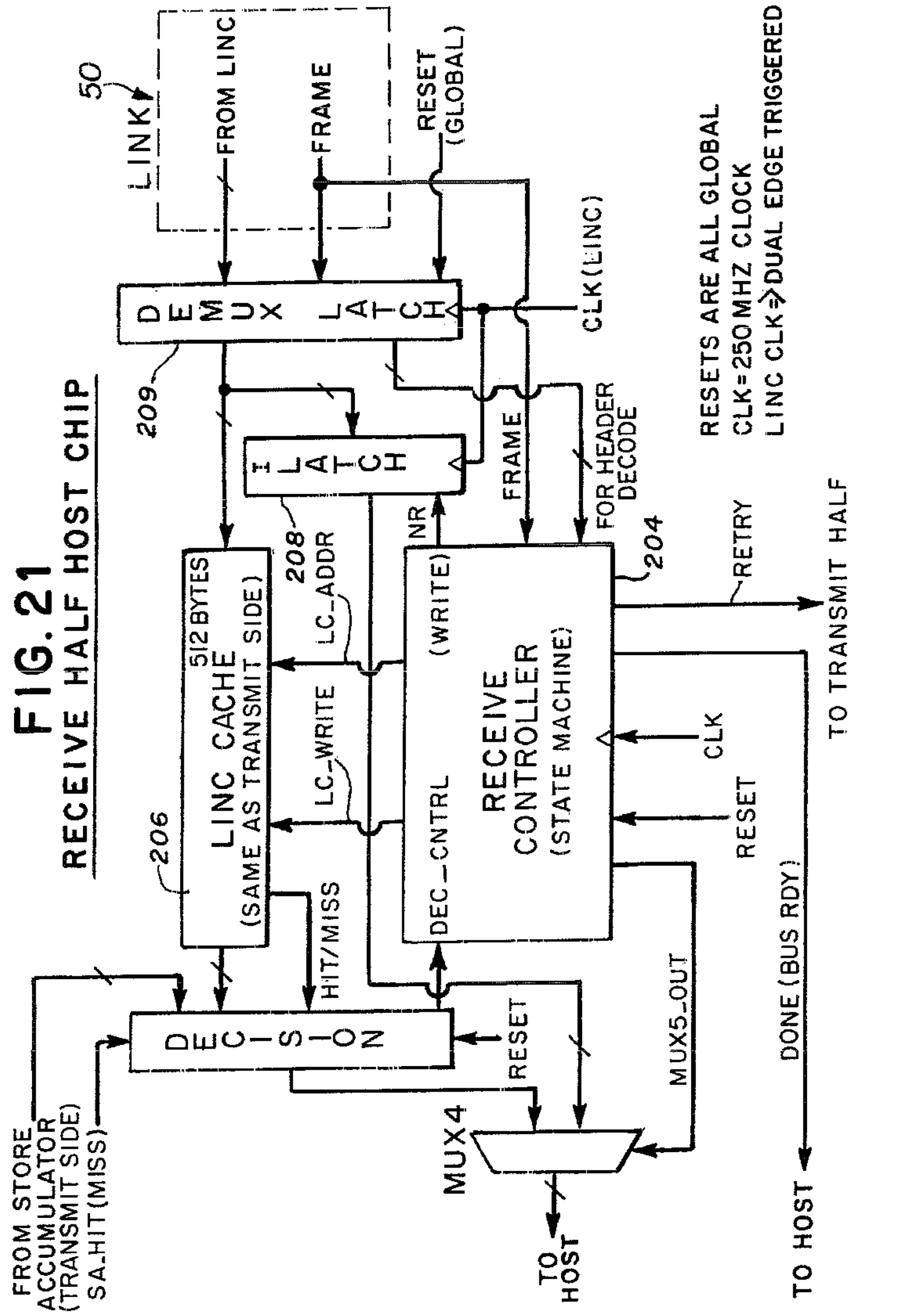

50
LINK
FROM LINC
FRAME
RESET (GLOBAL)
CLK (LINC)
DEMUX LATCH
209
I LATCH
208
FRAME
FOR HEADER DECODE
RESETS ARE ALL GLOBAL
CLK=250MHZ CLOCK
LINC CLK⇒DUAL EDGE TRIGGERED
512 BYTES
LINC CACHE
(SAME AS TRANSMIT SIDE)
206
LC_ADDR
LC_WRITE
NR
(WRITE)
DEC_CNTRL
RECEIVE CONTROLLER
(STATE MACHINE)
204
RETRY
TO TRANSMIT HALF
CLK
RESET
HIT/MISS
DECISION
RESET
MUX5_OUT
DONE (BUS RDY)
FROM STORE ACCUMULATOR (TRANSMIT SIDE)
SA_HIT(MISS)
MUX4
TO HOST
TO HOST

FIG. 21
RECEIVE HALF HOST CHIP

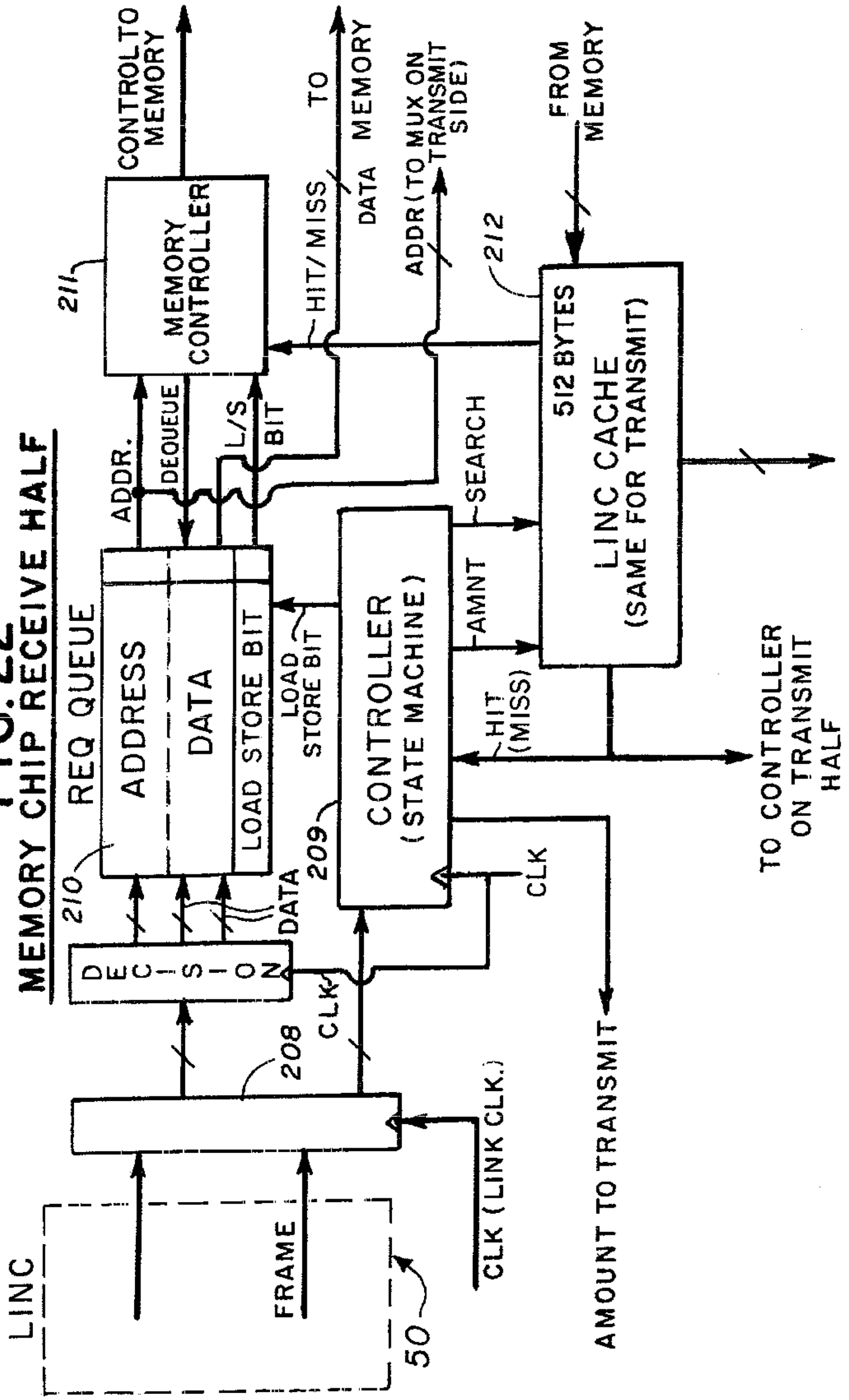

LINC
50
FRAME
CLK (LINK CLK.)
208
CLK
DECISION
DATA
210
REQ QUEUE
ADDRESS
DATA
LOAD STORE BIT
LOAD STORE BIT
209
CONTROLLER (STATE MACHINE)
CLK
AMOUNT TO TRANSMIT
ADDR.
DEQUEUE
L/S
BIT
211
MEMORY CONTROLLER
CONTROL TO MEMORY
HIT/MISS
DATA
TO MEMORY
ADDR (TO MUX ON TRANSMIT SIDE)
212
FROM MEMORY
SEARCH
AMNT
HIT (MISS)
512 BYTES
LINC CACHE (SAME FOR TRANSMIT)
TO CONTROLLER ON TRANSMIT HALF

FIG. 22
MEMORY CHIP RECEIVE HALF

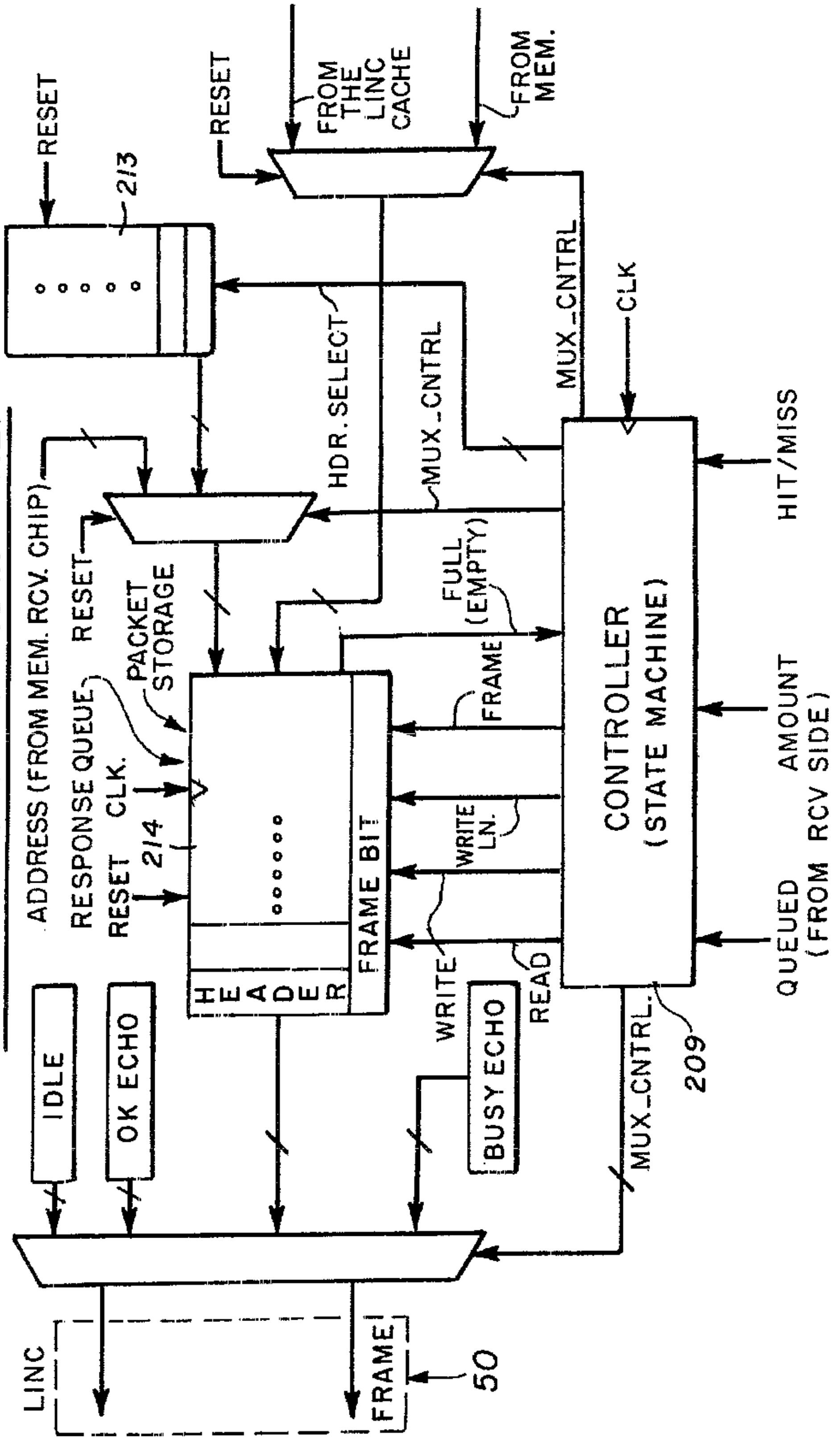

RESET
213
(HEADER POOL)
HEADERS
RESET
FROM THE LINC CACHE
FROM MEM.
MUX_CNTRL
HDR. SELECT
MUX_CNTRL
CLK
HIT/MISS
ADDRESS (FROM MEM. RCV. CHIP)
RESPONSE QUEUE
RESET
PACKET STORAGE
CLK.
214
RESET
FULL (EMPTY)
FRAME
FRAME
WRITE LN.
WRITE
READ
FRAME BIT
HEADER
CONTROLLER (STATE MACHINE)
AMOUNT
QUEUED (FROM RCV SIDE)
IDLE
OK ECHO
BUSY ECHO
MUX_CNTRL.
209
LINC
FRAME
50

FIG. 23
MEMORY CHIP TRANSMIT HALF

FIG. 24

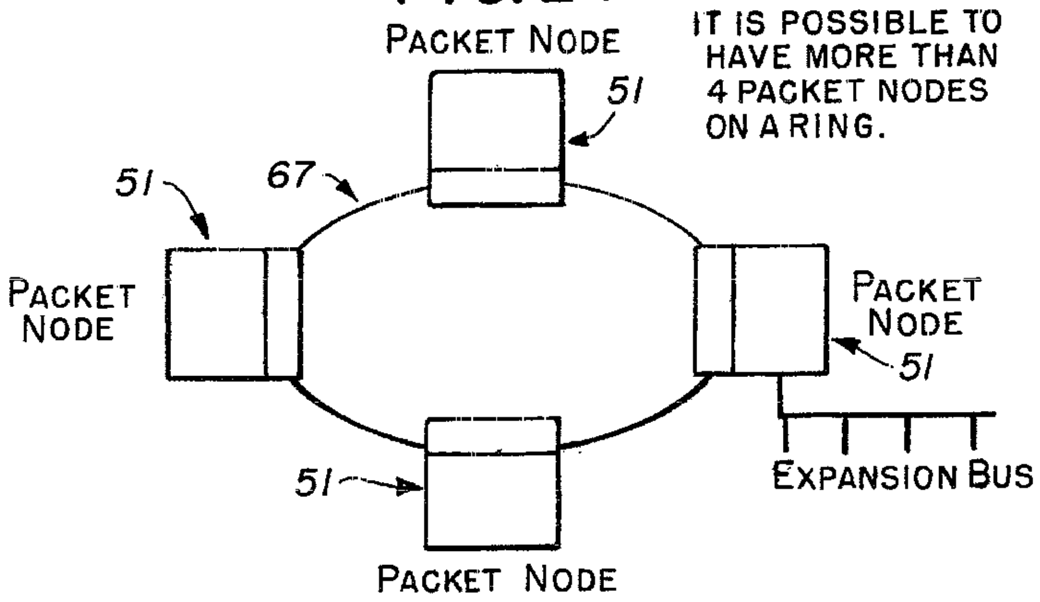

PACKET NODE
IT IS POSSIBLE TO HAVE MORE THAN 4 PACKET NODES ON A RING.
51
67
51
PACKET NODE
PACKET NODE
51
51
EXPANSION BUS
PACKET NODE

FIG. 25

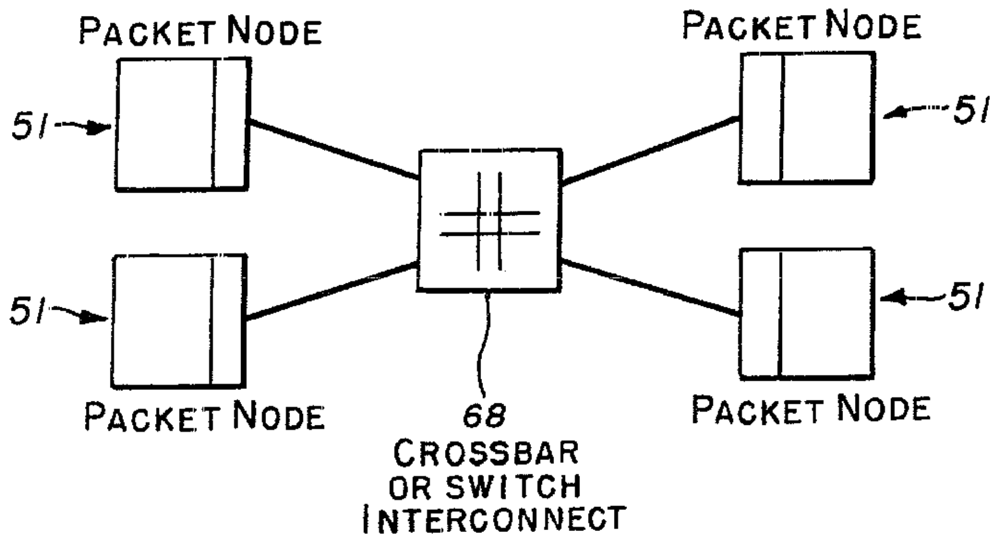

PACKET NODE
PACKET NODE
51
51
51
51
PACKET NODE
68
CROSSBAR OR SWITCH INTERCONNECT
PACKET NODE

FIG. 26

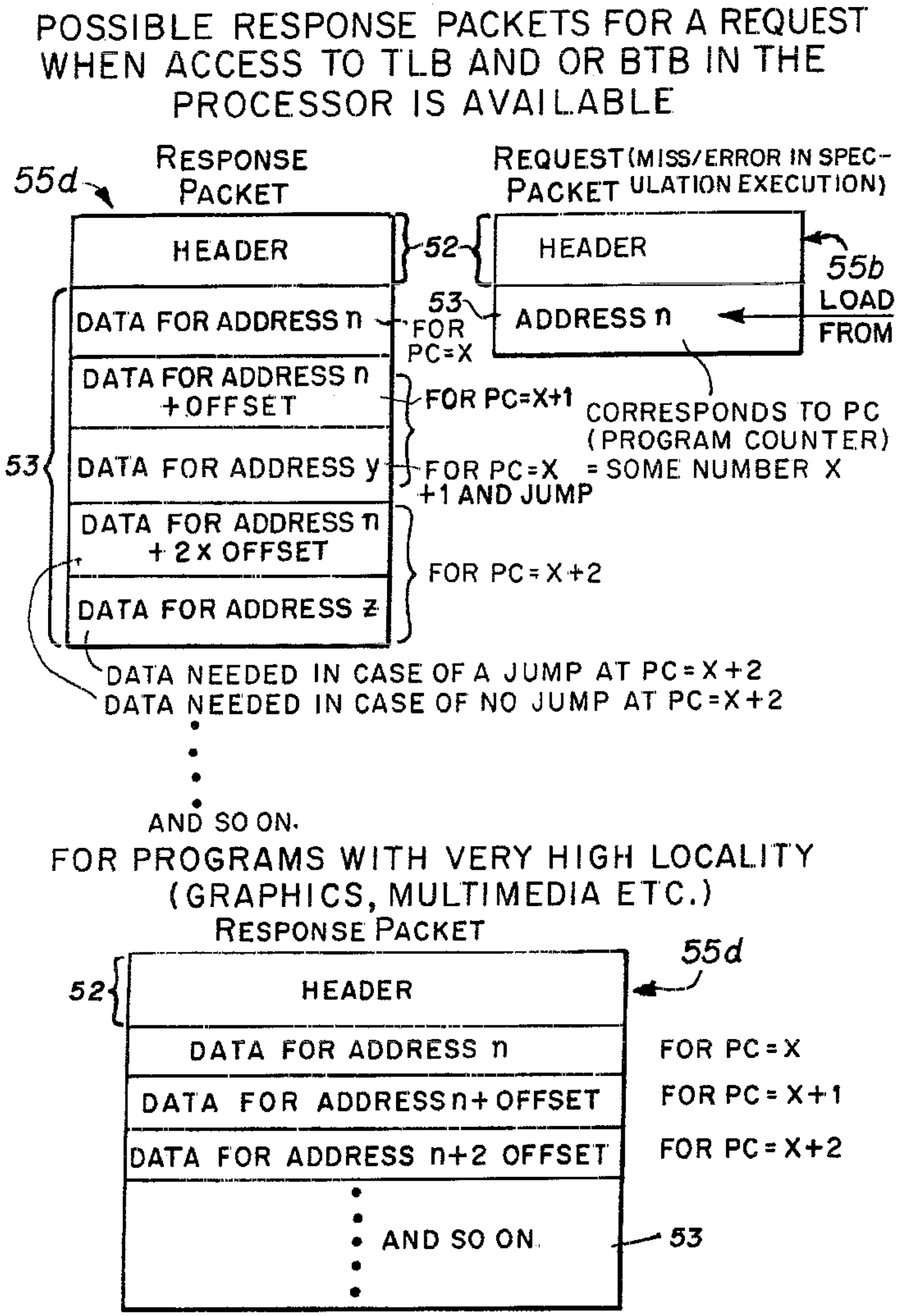

POSSIBLE RESPONSE PACKETS FOR A REQUEST WHEN ACCESS TO TLB AND OR BTB IN THE PROCESSOR IS AVAILABLE
55d
RESPONSE PACKET
REQUEST PACKET (MISS/ERROR IN SPECULATION EXECUTION)
HEADER
52
HEADER
55b
LOAD FROM
53
ADDRESS n
DATA FOR ADDRESS n
FOR PC=X
DATA FOR ADDRESS n +OFFSET
FOR PC=X+1
CORRESPONDS TO PC (PROGRAM COUNTER) = SOME NUMBER X
53
DATA FOR ADDRESS y
FOR PC=X +1 AND JUMP
DATA FOR ADDRESS n + 2 x OFFSET
FOR PC=X+2
DATA FOR ADDRESS z
DATA NEEDED IN CASE OF A JUMP AT PC=X+2
DATA NEEDED IN CASE OF NO JUMP AT PC=X+2
AND SO ON.
FOR PROGRAMS WITH VERY HIGH LOCALITY (GRAPHICS, MULTIMEDIA ETC.)
RESPONSE PACKET
55d
52
HEADER
DATA FOR ADDRESS n
FOR PC=X
DATA FOR ADDRESS n+OFFSET
FOR PC=X+1
DATA FOR ADDRESS n+2 OFFSET
FOR PC=X+2
AND SO ON
53

FIG. 27

PROCESSOR NODE RECEIVE PROTOCOL

FOR PRESENT DETAILED IMPLEMENTATION, THE MEMORY NODE IS EXPECTED TO SEND ONLY THE FOLLOWING PACKETS.

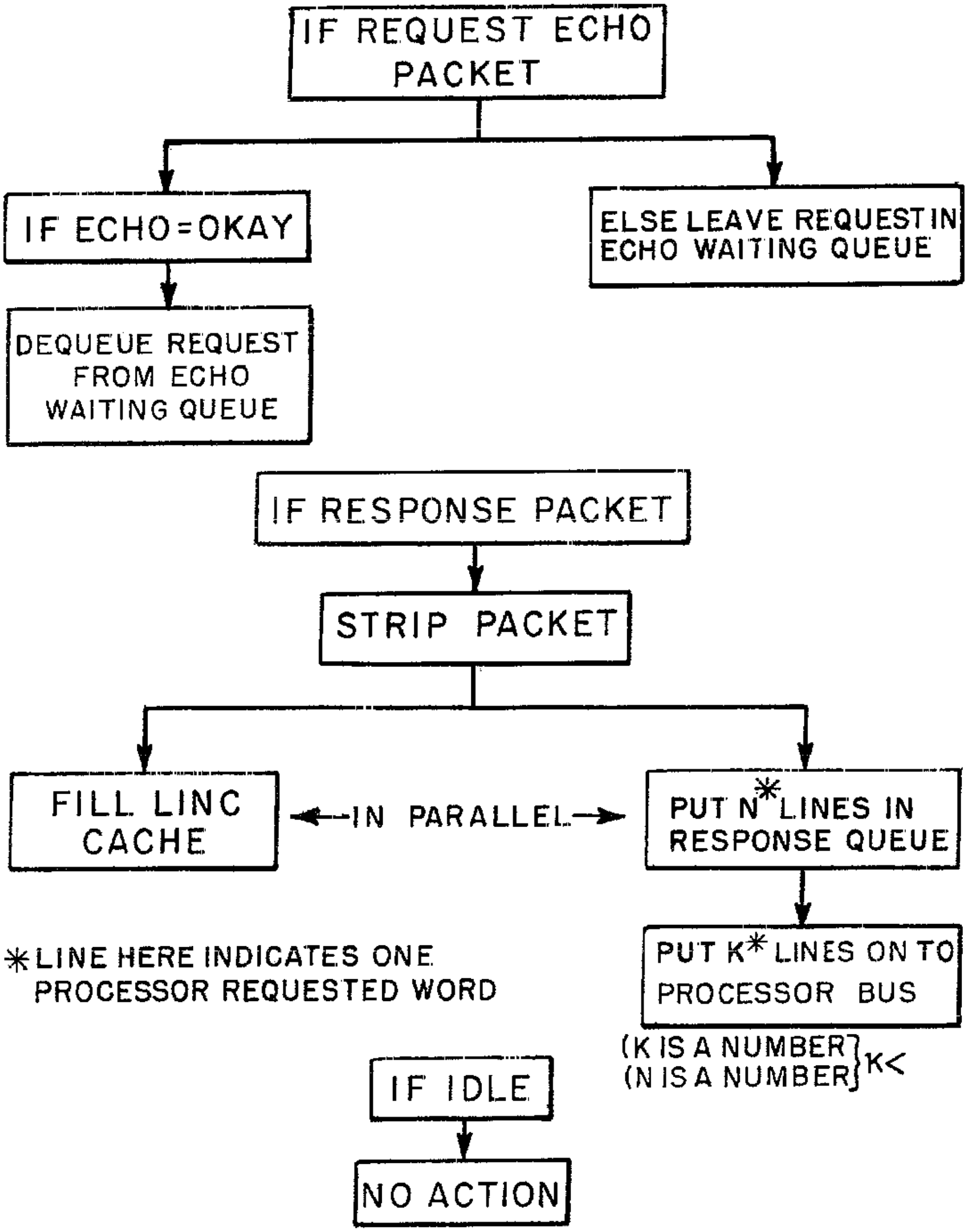

IF REQUEST ECHO PACKET
IF ECHO=OKAY
ELSE LEAVE REQUEST IN ECHO WAITING QUEUE
DEQUEUE REQUEST FROM ECHO WAITING QUEUE
IF RESPONSE PACKET
STRIP PACKET
FILL LINC CACHE
IN PARALLEL
PUT N* LINES IN RESPONSE QUEUE
PUT K* LINES ON TO PROCESSOR BUS
*LINE HERE INDICATES ONE PROCESSOR REQUESTED WORD
(K IS A NUMBER
(N IS A NUMBER
K<
IF IDLE
NO ACTION

FIG. 28

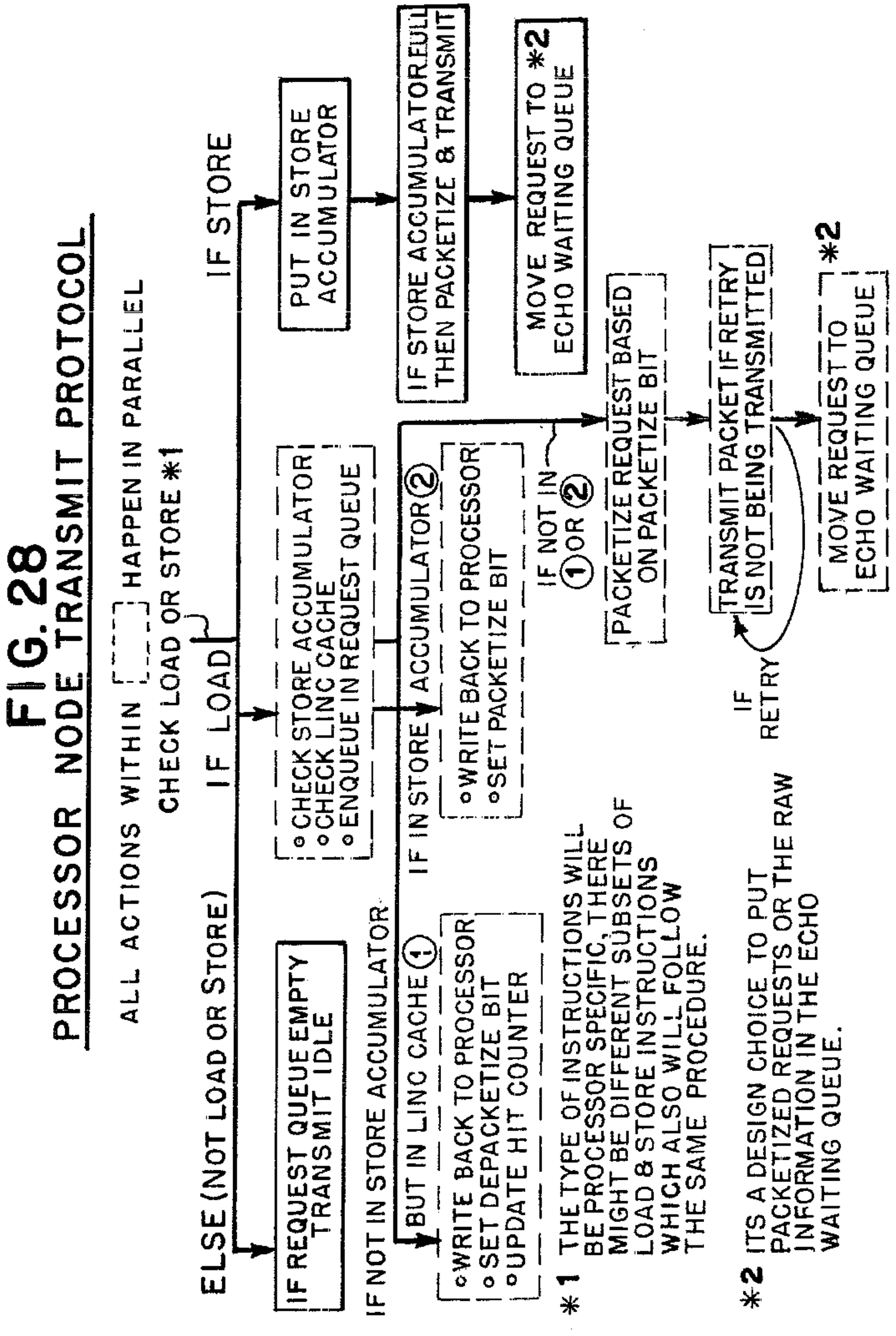

PROCESSOR NODE TRANSMIT PROTOCOL
ALL ACTIONS WITHIN [ ] HAPPEN IN PARALLEL
CHECK LOAD OR STORE *1
ELSE (NOT LOAD OR STORE)
IF LOAD
IF STORE
IF REQUEST QUEUE EMPTY TRANSMIT IDLE
∘ CHECK STORE ACCUMULATOR
∘ CHECK LINC CACHE
∘ ENQUEUE IN REQUEST QUEUE
PUT IN STORE ACCUMULATOR
IF STORE ACCUMULATOR FULL THEN PACKETIZE & TRANSMIT
MOVE REQUEST TO *2 ECHO WAITING QUEUE
IF NOT IN STORE ACCUMULATOR
BUT IN LINC CACHE ①
∘ WRITE BACK TO PROCESSOR
∘ SET DEPACKETIZE BIT
∘ UPDATE HIT COUNTER
IF IN STORE ACCUMULATOR ②
∘ WRITE BACK TO PROCESSOR
∘ SET PACKETIZE BIT
IF NOT IN ① OR ②
PACKETIZE REQUEST BASED ON PACKETIZE BIT
TRANSMIT PACKET IF RETRY IS NOT BEING TRANSMITTED
IF RETRY
MOVE REQUEST TO ECHO WAITING QUEUE *2
*1 THE TYPE OF INSTRUCTIONS WILL BE PROCESSOR SPECIFIC, THERE MIGHT BE DIFFERENT SUBSETS OF LOAD & STORE INSTRUCTIONS WHICH ALSO WILL FOLLOW THE SAME PROCEDURE.
*2 ITS A DESIGN CHOICE TO PUT PACKETIZED REQUESTS OR THE RAW INFORMATION IN THE ECHO WAITING QUEUE.

FIG. 29

MEMORY NODE RECEIVE PROTOCOL

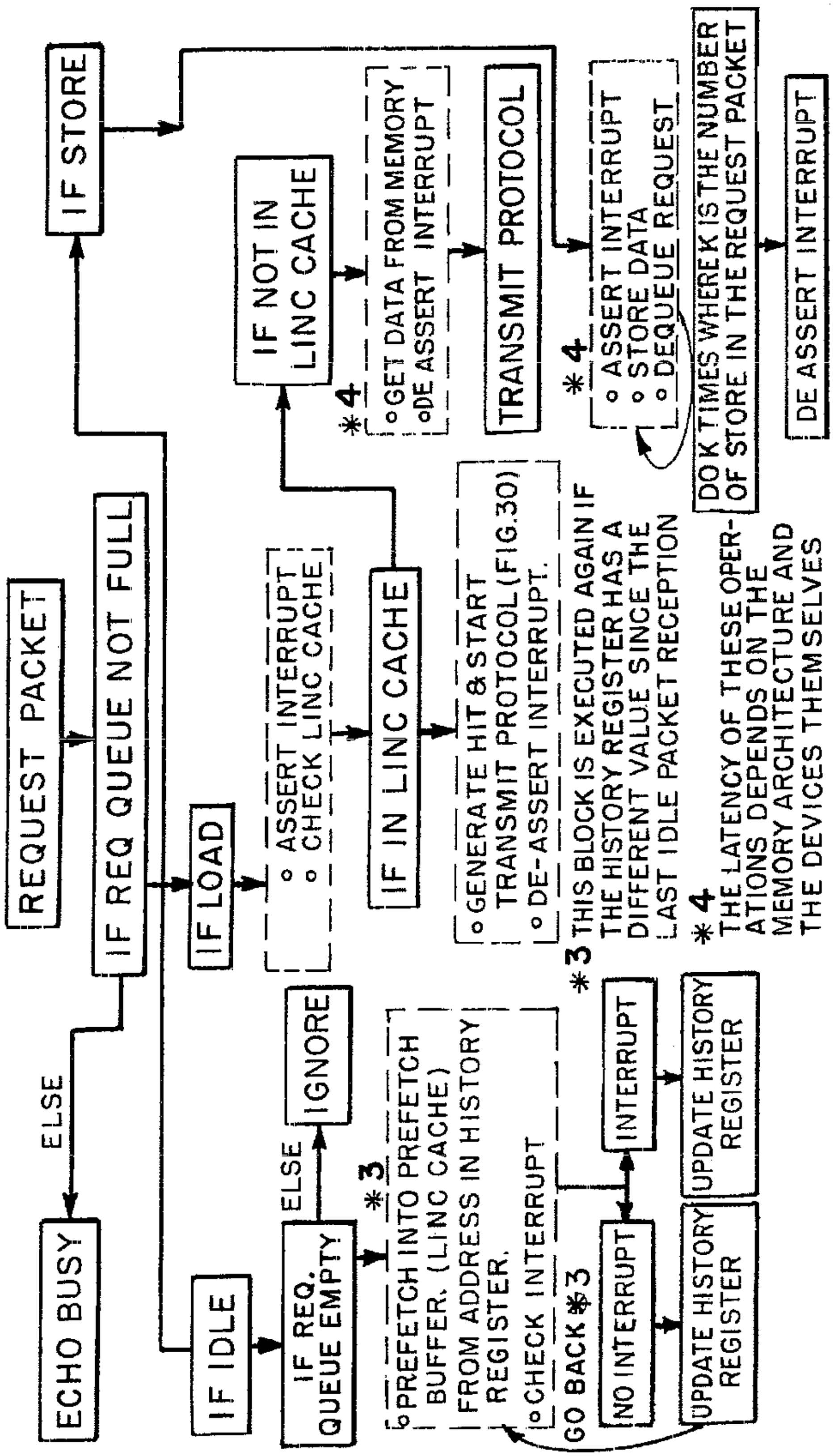
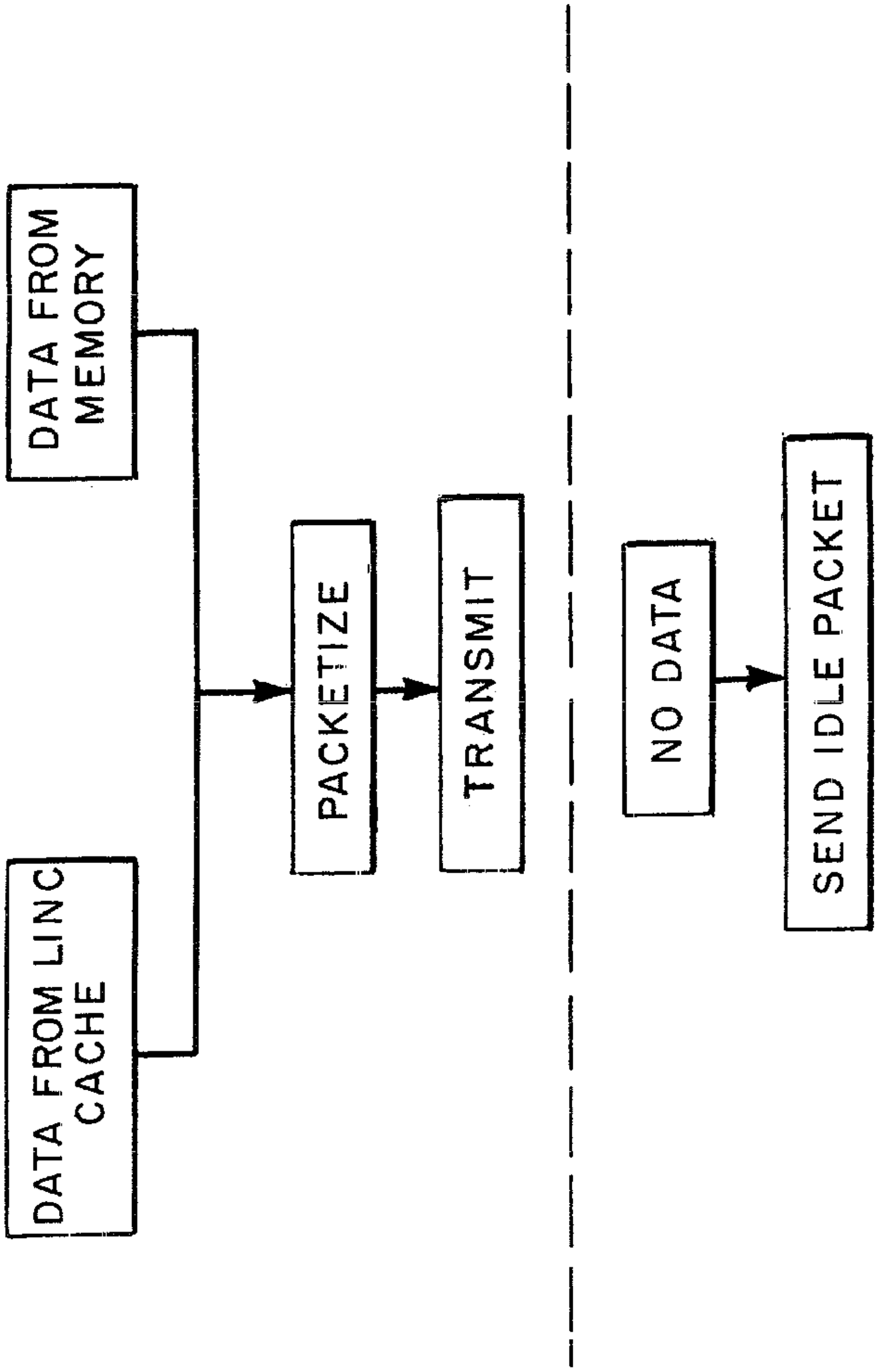

REQUEST PACKET
IF REQ QUEUE NOT FULL
ELSE
ECHO BUSY
IF STORE
IF LOAD
IF IDLE
o ASSERT INTERRUPT
o CHECK LINC CACHE
IF IN LINC CACHE
IF NOT IN LINC CACHE
*4
o GET DATA FROM MEMORY
o DE ASSERT INTERRUPT
TRANSMIT PROTOCOL
o GENERATE HIT & START TRANSMIT PROTOCOL (FIG.30)
o DE-ASSERT INTERRUPT.
*4
o ASSERT INTERRUPT
o STORE DATA
o DEQUEUE REQUEST
DO K TIMES WHERE K IS THE NUMBER OF STORE IN THE REQUEST PACKET
DE ASSERT INTERRUPT
IF REQ. QUEUE EMPTY
ELSE
IGNORE
*3
o PREFETCH INTO PREFETCH BUFFER. (LINC CACHE) FROM ADDRESS IN HISTORY REGISTER.
o CHECK INTERRUPT
GO BACK *3
NO INTERRUPT
INTERRUPT
UPDATE HISTORY REGISTER
UPDATE HISTORY REGISTER
*3 THIS BLOCK IS EXECUTED AGAIN IF THE HISTORY REGISTER HAS A DIFFERENT VALUE SINCE THE LAST IDLE PACKET RECEPTION
*4 THE LATENCY OF THESE OPERATIONS DEPENDS ON THE MEMORY ARCHITECTURE AND THE DEVICES THEMSELVES

FIG. 30

MEMORY NODE TRANSMIT PROTOCOL


DATA FROM LINC CACHE
DATA FROM MEMORY
PACKETIZE
TRANSMIT
NO DATA
SEND IDLE PACKET

COMPUTER HAVING PACKET-BASED INTERCONNECT CHANNEL

BACKGROUND OF THE INVENTION

This invention relates to physically non-distributed microprocessor based systems and, more particularly, the communication channel between functional components within a class of computer systems commonly known as personal computers

A personal computer (PC) is comprised of several major functional components which may be basically defined as a microprocessor, a read/write memory (RAM), a mass storage device (e g , hard drive or CD ROM), and an input/output (I/O) device (e g , display, serial port, parallel ports, etc) These functional components within the PC are interconnected by, and communicate via, a parallel data/address bus which is usually as wide as the processor data I/O path The bus is typically of fixed physical length comprising a number of parallel copper traces on the PC's motherboard In addition, there are provided a number of fixed tap points to the bus, e g , edge connectors, din connectors, etc , to allow the customization of the PC's configuration by adding peripheral functions, memory, etc , or removing unused functionality

While a bus provides a simple-minded mechanism for customization and communication within a PC, it has several limitations and unique problems associated with it First, a bus is by nature, single transaction (e g , only one functional unit can communicate with another at any given time and during this time, no other functional units can communicate with anything) and sequential (messages follow one after the other with considerable handshaking between functional units) A second problem of a bus is that all functional units connected to the bus must meet the electrical specifications and requirements of the bus even if these specifications and requirements are quite dated, technologically Thirdly, because the bus is a generic interconnect in nature, it can not be truly optimized for communication between any specific subset of functional units without adversely affecting communication performance between another subset of functional units

Fourth, the speed of the bus is substantially slower than might otherwise be obtainable This is due to two primary issues: First, busses are composed of relatively long lengths of parallel traces in close proximity to one another and this results in high parasitic capacitive coupling between traces of the bus (i e , electrical noise) This noise increases as the frequency, or speed, of the bus increases Thus, noise margin requirements restrict the speed (and length) of the bus The second issue relates to the unknown and highly variable electrical loading of the bus The speed of the bus is inversely proportional to the capacitive load on the bus This capacitive load is determined by the number of electrical connectors on the bus and the number of electrical connections to the bus. Since these numbers are variable, designers typically engineer the bus for worst case constraints That is, the bus is typically slowed down to a rate that would sustain a worst case loading situation even though this may occur in one PC in a thousand

Other major drawbacks of a bus are the need for electrical handshake signals and its fixed electrical data width (i e , 8 bits, 16 bits, 32 bits, etc) Handshake signals typically include READ, WRITE, MEM, I/O, WAIT/READY, etc. These signals are physical and are used to inform and control functional units (i e , inform of the type of request, and control/synchronize between communicators) Fixed data

width limitations become problematic as chip data path widths exceed the width of the bus. As will be seen herein, defining handshaking and data size at the physical layer is less flexible than would be desired

With the ever increasing demand for data manipulation in such applications as multimedia or graphics programs, the bottleneck of the bus becomes more acute There have been many attempts to address and remedy this problem (e.g., VESA, Video local bus, PCI, etc.) but no solution offers greatly improved performance and complete scalability

The present invention provides a system with the configuration flexibility of a bus-based PC while reducing the electrical problems Commensurably, interfunctional-unit communication speed and flexibility are greatly enhanced. The present invention applies a point-to-point packetized interconnection structure to facilitate communication between functional units (e g., processor, memory, disk, I/O, etc) within a PC

Because it is point-to-point, the interconnections scheme of the present invention is of relatively fixed electrical load and can, therefore, be optimized for speed Furthermore, the packet protocol that will be more fully disclosed herein provides a means of eliminating the typical physical layer control signals of a bus and replacing them with link-layer control which is much more flexible

In order to allow for interconnecting more than two functional units, the present invention may be expanded by any of several interconnect topologies, e g , switches, rings, etc Where speed and a high degree of parallel traffic is desired, a switch topology provides the best means, e.g , crossbar switch If speed is important but parallel traffic patterns are not very common, a shuffle-type switch may make the most sense In applications that are very cost sensitive, the present invention may also be expanded by means of a ring topology

As will be made clear in the specific disclosure portion of this document, the packetized point-to-point interconnection scheme of the present invention improves speed and performance at reduced cost and with better noise characteristics (both internal electrical noise and radiated EMI) as compared to the bus interconnect currently employed within a PC

Therefore, it is an object of the present invention to provide a new and improved PC, specifically, improving internal communication between microprocessor, memory, mass storage, I/O, etc or any subset of these functional units It is further an object of the invention to improve communication speed within a PC It is further an object of the invention to reduce interconnection electrical noise within a PC It is further and object of the invention to provide a more flexible interconnect means within a PC

Accordingly, it is a general object of the present invention to provide a new and improved PC, specifically improving internal communication between microprocessor, memory, mass storage, I/O, etc or any subset of these functional units

It is a more specific object of the present invention to provide improved communication speed within a PC

It is a still more specific object of the present invention to reduce interconnection electrical noise within a PC, and to provide a more flexible interconnect means

SUMMARY OF THE INVENTION

The invention is directed to a physically non-distributed microprocessor-based computer system, comprising a microprocessor, a random access memory device, a mass storage device, an input-output port device, wherein the devices are each being operable in conjunction with the microprocessor and include an interface for receiving and transmitting data in packet form, and which further comprise a packet-based data channel extending between the microprocessor and the interfaces of the devices for providing simultaneous bi-directional communication between the microprocessor and the devices.

BRIEF DESCRIPTION OF THE DRAWINGS

The features of the present invention which are believed to be novel are set forth with particularity in the appended claims. The invention, together with the further objects and advantages thereof, may best be understood by reference to the following description taken in conjunction with the accompanying drawings, in the several figures of which like reference numerals identify like elements, and in which:

FIG. 1 is a block diagram of current PC systems which use a bus based interconnect.

FIG. 2 is a conceptual block diagram of two packet nodes and a physical link.

FIG. 3 is a general block diagram of a packet.

FIG. 4 is a detailed diagram of the packet header showing each of the fields.

FIG. 5 is a general block diagram of an Idle packet.

FIG. 6 is a general block diagram of an Request packet.

FIG. 7 is a general block diagram of a Request Echo packet.

FIG. 8 is a general block diagram of a Response packet.

FIG. 9 is a functional block diagram of a link interface showing all the necessary elements.

FIGS. 10*a* and 10*b* show the possible structures of a request queue shown in FIG. 9.

FIGS. 11*a* and 11*b* show the possible structures of a response queue shown in FIG. 9.

FIG. 12 shows the structure of a line cache shown in FIG. 9.

FIGS. 13*a* and 13*b* show possible implementations of the line using discrete link interface chips.

FIG. 14 shows a processor with an embedded link interface connected to system memory through another link interface.

FIG. 15 shows detailed structure of the Idle packet for processor memory I/O.

FIG. 16 shows detailed structure of the request packet for processor memory I/O.

FIG. 17 shows a detailed structure of a Request Echo packet for processor memory I/O.

FIG. 18 shows detailed structure of a Response packet for processor memory I/O.

FIG. 19 shows sample request packets for a load and store instruction.

FIG. 20 is a detailed schematic of the processor node interface chip transmit half.

FIG. 21 is a detailed schematic of the processor node interface chip receive half.

FIG. 22 is a detailed schematic of the memory node interface chip receive half.

FIG. 23 is a detailed schematic of the memory node interface chip transmit half.

FIG. 24 shows a possible ring interconnect topology for packet nodes.

FIG. 25 shows a possible switched interconnect topology for packet nodes.

FIG. 26 shows possible structures of response packets for a given request in response to conditional branch or jump in program code.

FIG. 27 is a flow graph of the processor node receive protocol.

FIG. 28 is a flow graph of the processor node transmit protocol.

FIG. 29 is a flow graph of memory node receive protocol.

FIG. 30 is a flow graph of memory node transmit protocol.

DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference to the Figures, and particularly to FIG. 1, a node is defined as any device or group of devices which perform a specific system function within a microprocessor system. Such nodes typically have a physical address or addresses with which they are associated. Examples of such nodes include a processor, a memory, an input/output device, a hard disk, etc. The object of this invention is to interconnect all, or some subset of all, nodes with a high speed message passing packetized interconnection channel.

FIG. 1 depicts the simplest implementation of the prior art. Processor node **40** is connected to bridge node **41**. Bridge node **41** is further connected to memory node **42**. Bridge node **41** also provides an interface to system bus **43** in order to allow processor node **40** to communicate with peripheral nodes **44, 45, 46** and **47** over the bus.

The high speed message passing packetized interconnect channel and protocol is intended to allow asynchronous communication between any two nodes, so equipped, in any system configuration using packets of data or instructions. FIG. 2 shows a logical conceptualization of a node. The functional unit **48** may be any of the elements of FIG. 1. Interface **49**, which is the embodiment of the present invention, provides a seamless interface between functional unit **48** and the physical layer **50**. Physical layer **50** connects to any other packet node **51**, thereby, in conjunction with interface **49**, allowing communication between any group of functional units of FIG. 1.

Thus, the present invention makes each node in a computer microprocessor/microcontroller based system interface with each other using a uniform packed based message passing interface. The invention creates packet nodes which speak the same language electrically and logically. This simplifies communication and minimizes traditional handshaking and overhead. Therefore, significant data-rate speedup may be gained because rather than being pigeonholed into a hardware restricted, single transaction, high overhead interconnect (i.e., a bus), the present invention allows messages of varying length (rather than single transaction) with flexible handshaking, and minimal overhead. Or, more precisely, the present invention allows for more intelligent and flexible information exchange between nodes while requiring only that overhead and handshaking required for a specific transaction. The present invention does not impose, as does a bus, a set of costly rules and formalism that must be adhered to even if it makes no sense for a specific transaction.

Again, with reference to FIG. 1, current systems communicate at the hardware level, i.e., the processor node **40** issues commands on the processor bus which must then be converted from the virtual address that the processor understands to a physical address by bridge **41**. This, information is then further converted by bridge **41** prior to being placed on the bus **43** At this point, all other nodes, **42**, **44**, **45**, **46** or **47** on the bus look to see if they are being addressed by processor node **40**. The one node that is in fact being addressed, acknowledges the processor query and then takes action based on that query Upon receiving the acknowledgment from the queried node, processor node **40** now may take appropriate data action (i.e , output data, input data, etc) and the transaction is assumed to be completed Next, the process can repeat in exactly the same manner, even if the query is to the exact same node as previously addressed This is the handshaking and overhead bottleneck of a conventional bus. Furthermore, whenever the bus is being utilized by a pair of nodes, all other nodes are prevented from communicating with any other node The present invention stems from the realization that the byte by byte handshaking and transmission that is typical of a bus-based system is greatly inefficient and constraining and may be vastly improved upon.

In accordance with the invention packet based message passing techniques are applied to the nodes within a microprocessor based system With reference to FIG **2**, the following is a simple illustration of the invention's operation: Execution unit **48***a* (e.g , processor node) requests information from execution unit **48**(*b*) (e.g., memory node) via link interface **49**(*a*) Link interface **49**(*a*) assembles a packet requesting said information (i.e., its address in memory, amount of data requested, etc) and then rapidly transmits said packet to link interface **49**(*b*) via physical interface **50** (which may be single-ended line drivers, low voltage differential drivers, or any other method common in the art) Link interface **49**(*b*) then decodes the packet and takes the necessary steps to process the request with respect to execution unit **48**(*b*). Link interface **49**(*b*) then collects, from execution unit **48**(*b*), all data necessary to fill the request, packetizes the data into a response, and then, when the request is filled, ships the data back to link interface **49**(*a*) via physical link **50** At this point, link interface **49**(*a*), depacketizes the data and provides it to execution unit **48**(*a*) in a manner befitting the execution unit's request Thus, the physical link is only tied up for the time when useful information is actually being sent Furthermore, requests for several pieces of data result in less physical interface bandwidth utilization since the several pieces of data are streamed in the same message

A packet node in the present invention communicates with another packet node using "packets. Each packet contains all the necessary information that is required for the intended receiver without the added overhead of setting up the receiver or formatting the data to a node-specific set up

The structure of a packet **55** such as shown in FIG **3** may have, for example, the following general characteristics:

1) Packet components **52** and **53** are of 16 bits (2 bytes) in width and referred to as a packet word

2) Header **52** has means for indicating that the next packet word is an extension of the header **52** This means is the extended header bit **54**.

3) A packet body **53** which can be anywhere from 0 bytes to 256 bytes in length.

4) The maximum size of a packet **55** is 258 bytes.

5) The width of the packet remains the same regardless of the width of the channel (i.e., for wider channels, more than one packet word may be sent in parallel)

The above definitions may easily be changed without affecting the nature of the invention.

FIG. **4** details the fields within the header packet The type-of-packet field **59** defines one of four fundamental

packet types that are exchanged between any two nodes. These are 1) IDLE, 2) REQUEST, 3) REQUEST ECHO, and 4) RESPONSE The type-of-instruction field **58** indicates the action that needs to be taken by the receiving node of this packet Examples of such actions include load, store, input, output, read, write, and other system level interfunctional unit operations

The size-of-device field **57** is to allow for the interface between devices of different physical data widths with minimum physical layer transmission time By knowing the size of a requesting node device, the interface circuitry of the receiving node can pack the data into a packet in the most efficient manner for decoding by the requesting device and only send portions of the overall required response that are filled, where 'filled' is defined as sufficient to meet the width of the requesting node device as defined in the size-of-device field

The flow control field **56** contains the size-of-response, node ID, extended header bit, and BUSY/OKAY status bit The size-of-response indicates the amount of data being requested The node ID indicates the logical functional unit for which this packet is intended. The extended header bit allows for headers greater than 2 bytes in size, where necessary The BUSY/OKAY status bit indicates whether the receiver of request packet can accept and service the packet.

Having defined the fields of the header packet, we now define the four fundamental packet types The IDLE packet **55***a*, FIG. **5**, contains only the header and no data and is continually sent out by the idle node A node receiving the IDLE packet may then use the idle link for transmitting data that the receiving node believes the idle node may need based on the idle node's prior request history.

It is informative to illustrate the use of IDLE packets with an example Assume that the processor node has been requesting sequential data blocks from the memory node. At some point in time, the processor node stops requesting data from the memory node because the processor node has to do something else (e.g., service an interrupt). At this point, the processor node sends idle packets to the memory node Upon reception of the idle packet, the memory node reviews the history of processor node requests and may continue to send data based on a projection of the history of the processor node requests These unrequested data are then stored in the processor node link interface cache provided the processor node has not specifically requested some data from any other node In this way, idle links can be used most effectively to transmit data that may be needed before it is requested

A second type of packet is the REQUEST packet **55***b*, FIG **6**, which has been informally referred to throughout this disclosure This packet is transmitted between any two nodes to indicate or request an action from the receiving node for the requesting node The request packet contains a header (see FIG **3**, element **52**) that has the ID of the requested node in the flow control field and the type of instruction for the receiver to execute, e.g , load, store, etc. The request packet also contains a body (see FIG **3**, element **53**) to the extent that there is data sent by the requesting node to the receiving node for the receiver to perform the requested instruction.

The REQUEST ECHO packet **55***c*, FIG **7**, is sent by the receiving node to acknowledge reception of a REQUEST packet **55***b* This packet is primarily for indicating whether the request from the requester can be catered or not Within the header of the request echo packet, in the flow control field, the REQUEST ECHO packet indicate whether the receiver is busy (busy echo) or able to service the request (okay echo)

The last packet type is the RESPONSE packet **55***d*, FIG. **8** This packet is used to respond to a REQUEST packet The header of the RESPONSE packet contains the node ID of the intended receiver (i.e., the original requesting node) and other information regarding flow control, etc. The body of the response packet contains the data requested to the extent it is required and the body of the RESPONSE packet is no longer than it needs to be to hold said data

A typical transaction between any two nodes (node a and node b) shown in FIG 2 is summarized below:

1 Node A generates a request packet for Node B

2 Node B, based on whether Node B's request queue can cater to the request, sends one of two messages back:
   a If it can cater to request from Node A then sends Node B a REQUEST ECHO OKAY packet
   b If it can not cater to a request from Node A then Node B sends a REQUEST ECHO BUSY packet

3 If Node A receives a REQUEST ECHO OKAY packet then Node A takes no action on the original request. If Node A receives a REQUEST ECHO BUSY packet then Node A resends the original request.

4 If REQUEST ECHO OKAY was sent by Node B, then Node B sends a RESPONSE packet to cater to the original request. This completes the transaction between Node A and Node B.

If no action is required from either Node A by Node B or from Node B by Node A, then IDLE packets are exchanged between them. Node A sending an IDLE packet to Node B or vice a versa are both independent operations The IDLE packet may also be used to exchange configuration/status/control information of each node.

To extend the capability of the present invention to an arbitrary number of nodes requires an expanded interconnect FIG **24** and FIG **25** depict two possible interconnection schemes

FIG. **24** shows a topology commonly referred to as a ring interconnect **67**. In this type of interconnect, each link interface's physical link output is connected to a neighboring nodes link interface physical input until the ring is closed For this type of implementation, the link interface must implement a pass-through mechanism That is, each link interface must compare its node ID to the node ID of the packet header If the compare is not successful, the link interface must forward the packet just received on its physical link input to its physical link output In this way, packets circulate in the ring until they arrive at their ultimate destination This receive-check-forward mechanism is functionally similar to that described in the IEEE 1596 (SCI) specification (Elastic Buffer)

To improve performance over a ring, FIG **25** shows another common interconnect topology commonly referred to as a switch **68** The switch **68** of FIG. **25** may be a crossbar switch, a shuffle switch, a broadcast crossbar switch, or similar device Implementations of crossbar switches are well known to the art and it is sufficient to describe a cross bar as N, M to 1 multiplexers, where N is the number of output ports and M is the number of input ports When a crossbar switch is used, each link interface must check the node ID of the received packet to guarantee that the packet is intended for the receiving node. This straightforward modification to the link interface physical link input circuitry is to include an ID decoder in the receive logic before queuing the request.

FIG. 9 depicts a functional diagram of the link interface 49. The Host Interface **60** provides the means to connect the link interface to the bus of the functional unit **48** (i.e., processor, memory, I/O, disk, etc.) This part of the link interface contains all the necessary hardware to handshake with the functional unit node and is specific to the said functional unit It also provides for all necessary signals to complete bus cycles needed for the functional unit

The Store Accumulator **61** is responsible for packing data into a packet body for the STORE instruction. This is especially useful when the processor node is doing a burst write. In this case, the several data and addresses that are sequentially output by the processor are accumulated by the Store Accumulator **61** of the link interface and packed into one store message packet. Thus, a single message transaction results in several data being stored by the receiving node

The Control block **62** provides for control of the internal components of FIG 9 as well as coordinating the functioning of the physical link. Control block **62** is essentially a state machine that keeps track of the link state and provides the necessary housekeeping functions of the link Control block **62** also contains the history register which is used by the requester in conjunction with the Line Cache **63** 'hit' information to determine the desired size-of-response for a given request This same register is used by the receiver to determine how much data and from where said data may be returned when the receiver detects an IDLE packet The detailed operations that Control block **62** performs are disclosed in association with the operation of each the blocks of FIG 9 and the link interface

Request Queue **64** provides buffering and storage for all accesses coming from a functional unit **48** These accesses can either be stored in raw form (node address and data format), or in packetized form, depending upon the access arrival rate. That is, based on the rate that accesses come into the link interface from the functional unit **48**, the access may be stored raw and then packetized as the access is converted to a request packet and placed on the physical link or the access may be packetized prior to being placed in the queue With reference to FIG. **10**, if the access is being stored as a packet in the Request Queue **64**, the queue is configured, as in FIG **10***a*, to be two bytes wide plus one bit for frame The frame bit **67** is used to indicate the presence of header on the current cycle If the access is being stored raw, then the request queue is reconfigured to be as wide as the functional unit's address width **68** plus data width **69** plus the instruction field width **70** as in FIG **10***b*

The Echo Waiting Queue **65** of FIG 9, is operable to function as storage for outstanding Request packets These Request packets are copied into the Echo Waiting Queue **65** until an ECHO OKAY packet is received from the node catering to the request. Storing outstanding requests provides the means for the Control Block to handle out-of-order RESPONSE packets and to verify link integrity by making sure that all requests are being responded to

The Response Queue **66** has a structure similar to the Request Queue **64** It may be configured as in FIG 11 and its operation is the reverse of the Request Queue. The Response Queue can store either packetized information as received from the physical link for later depacketization and passage to the functional unit (function unit data-need is slow), FIG **11***a*, or information received from the physical link may be immediately depacketized and stored raw for delivery to the functional unit (functional unit data-need is fast), as in FIG. **11***b*

Line Cache **63** is basically a directed mapped cache for caching response data for the functional unit The size of the Line Cache **63** is an integer multiple of the maximum data packet size **53**, i.e., m×256 To keep track of the latest data, the Line Cache **63** is partitioned into two identical blocks;

one block containing the latest information **71** and the other block containing the information received before and up to the latest update **72**.

With reference to FIG. **12**, each block of the Linc Cache **63** is of fixed size (width **73** and depth **74**). However, the Linc Cache line **75** size is variable. Furthermore, the Linc Cache line size is always at least as long as the functional unit's cache line **76** size. There are N words in a Linc Cache line, where N is dynamically variable. If an access from the functional unit misses in the Linc Cache, the link interface will request the data from the proper node. The amount of data requested by the link interface, N, depends on the history of the 'hit' rate within the Linc Cache. If the hit rate is high, the control circuitry increase the Linc Cache line size thus maximizing data transmission per physical link transaction. If the hit rate is low in the Linc Cache, the Linc Cache line size is reduced in order to reduce the size of messages on the physical link.

The motivation for this unique and counterintuitive approach to cache management is the realization that if the hit rate to the Linc Cache is low, the accesses are almost certainly not sequential and are unpredictable. Thus, increasing the Link Cache line size will probably not improve the hit rate. Therefore, the invention reduces the line size (which will probably not hurt the hit rate but will make the link available for all nodes more often since message sizes from this node will now be smaller).

FIGS. **13** and **14** depict two possible embodiments of the link interface **49**. FIG. **13** shows an implementation wherein the link interface is separate from any of the node's circuitry. FIG. **13***a* shows an implementation where the physical link is on a motherboard or external physical channel with link chips for each node. The link chips in FIG. **13***a* only contain the link interface **49**. FIG. **13***b* shows an implementation where the physical link is embedded inside a single chip or a linc chip **39**. The linc chip **39** contains both a link interface **49** and the physical link or channel **50**. FIG. **14** shows an implementation wherein the link interface associated with the processor node is include within the processor silicon itself. FIG. **13***b* will be discussed first.

FIGS. **20** through **23** show a detailed implementation of a link interface. With no loss of generality, the interconnect is assumed to be a two point, point to point interconnection between a processor node and a memory node. It will be shown in the description to follow how the memory node interface may be extended to include any peripheral device.

FIGS. **20** and **21** depict the transmit and receive, respectively, link interface circuitry for the processor node. With respect to FIG. **20**, the transmit operation starts when the processor node (host) begins a bus cycle. The transmit circuitry latches the necessary data and address information into latch **201**. As the data and address are being latched, the bus cycle of the processor node is decoded to be either a read or write (load or store). In either case, control circuitry **204** enables the appropriate header from header pool **207**. This header pool encodes all possible header types since the type of processor is known to the specific implementation of the link interface and allows for faster assembly of a packet. The header is put in the request queue **202** and then the address/data/control of the bus cycle is mapped to the packet as it is put in the request queue **202**. The request queue **202** operates in a first in, first out (FIFO) manner. If the cycle is deciphered to be a store then the request is stored in the store accumulator **205**. After the store accumulator **205** is full the control attaches a header to the information in the store accumulator **208** and sends it out on the physical linc **50**. Once any request has been sent out, it is queued in the echo waiting queue **203** to await the receipt of an ECHO OKAY. During the queuing of a request, when a load or read request is received from the processor node, there is a search done in the linc cache **206** and the store accumulator **205** to determine whether the request can be catered to without going out on the linc (i.e., check to see if a linc cache hit occurs).

Referring to FIG. **21**, in the receive operation, once a packet is received over the physical link, the intelligent latch **208** and demultiplexer **209** combination allows depacketization of the incoming information and the storing of the data/address in the linc cache **206**. Because of the variable linc cache line size, the returning requested data will be more than the original processor required amount (to fill the larger linc cache line). Finally, the control **204** issues the appropriate bus and control signals to satisfy the processor node request.

The receive operation for the memory node with respect to FIG. **22** is described below. The data comes in on physical link **50** and latch **208** takes the data and appropriately fills the request queue **210**. The request queue **210** contains the raw request which then goes to the memory controller **211** for the appropriate action. The design and implementation of the memory controller is specific to memory devices being used and the system memory architecture. Once a load request has been received, the controller **209** checks the linc cache **212** to see if the data is ready to be packetized and sent across the physical link **50**. This saves cycles since it is not necessary to assert the appropriate control signals to start the memory access cycle if there is a hit in the linc cache **212**. If the request received is a store request, then it is directly sent to the memory controller **211** for appropriate action.

The transmit operation with respect to FIG. **23** is briefly described below. Once a request is received, header pool **213** is indexed to provide the appropriate header for the response packet. The request that was received over physical link **50** and stored in the request queue **210** is dequeued and the response is then taken out of the linc cache **212** or the memory and packetized by attaching the appropriate headers and put into the response queue **214**. The response queue has a FIFO operation similar to any of the other queues used in the current implementation and so the response will be sent when it is at the head of the queue.

It is important to realize that FIGS. **22** and **23** describe an Functional Unit specific implementation embodying a memory node. With the addition of a store accumulator and the appropriate controller that replaces the memory controller **211** in FIG. **22**, this circuitry can be adapted to Functional Units of any type on any node.

The protocols used for communication between the processor node and the memory node are described in the flow graphs in FIGS. **27**, **28**, **29** and **30**. These protocols are:

1) Processor node Protocols
   a) Receive Protocol (FIG. **27**)
   b) Transmit Protocol (FIG. **28**) and
2) Memory node Protocols
   a) Receive Protocol (FIG. **29**)
   b) Transmit Protocol (FIG. **30**)

The flow graphs in conjunction with the description described herein describe the functioning of the link between the processor node and the memory node.

FIG. **14** depicts a provision of the link interface on the same silicon as the processor. This results in several important simplifications and improvements. Moving inside the processor silicon allows the link interface access to the Translation Lookaside Buffer (TLB) and the Branch Target Buffer (BTB) which allows the implementation of sophisticated prefetching and caching schemes.

Microprocessors typically do speculative execution based on the load/store instructions in a program. For standard arithmetic logic unit (ALU) operations, it is relatively easy to identify the register operands needed to be accessed during the instruction decode phase itself. However, for memory access operations, significant improvement is possible. In particular, the determination of the memory location that needs to be accessed requires an address calculation. The load/store instructions are issued to a pre-execute engine where address calculation is performed. After address calculation, the virtual address is translated into a physical address, if necessary. This address is then issued to the memory interface of the processor to form the appropriate request.

By moving the link interface inside the processor, access is gained to TLB which stores a lookup table or cache translation descriptors of recently accessed pages. This information is very valuable in doing intelligent memory prefetches because now the link interface can look at the TLB and decide the location and access size of prefetches.

When the link interface is within the processor, the interface also gains access to the BTB. This allows two important benefits. First, in any given program or code, there typically is a branch or jump every five instructions, on the average. The branch prediction mechanism of the processor allows the processor to do speculative execution by predicting where it needs to go four or five instructions ahead of the current program counter (PC). When the branch predictor is wrong, there is a huge performance penalty in the processor because of stalls and pipelines running empty. By having access to the BTB, the link interface knows all possible outcomes of the branch and can prefetch data/instructions for all of the possible outcomes and have this information available at the processor. In this way, performance penalties due to branch prediction errors are significantly reduced. This implements a virtual zero wait-state operation to memory on a branch miss.

Second, in the rare event that there are no jumps in the program, FIG. 26 shows a method of obtaining performance improvement. Rather than fetch a large block of data from a single address, each packet would be configured to fetch several smaller blocks of data from several different addresses. In this context, "data" means both program data and processor instructions.

For all of the above detailed embodiments of the invention, FIGS. 15, 16, 17, 18 and 19 show the specific implementations of the packet structures along with which fields of the header are active for each packet type.

While a particular embodiment of the invention has been shown and described, it will be obvious to those skilled in the art that changes and modifications may be made therein without departing from the invention in its broader aspects, and, therefore, the aim in the appended claims is to cover all such changes and modifications as fall within the true spirit and scope of the invention.

I claim:

1. A physically non-distributed microprocessor-based computer system, comprising:

a microprocessor;

a random access memory device;

a mass storage device;

an input-output port device;

said devices each being operable in conjunction with said microprocessor and including an interface for receiving and transmitting data in packet form; and

a packet-based data channel extending between said microprocessor and said interfaces of said devices for providing simultaneous bi-directional communication between said microprocessor and said devices.

2. A computer system as defined in claim 1 wherein said data channel comprises a first signal path in one direction and a second signal path in the other direction.

3. A computer system as defined in claim 1 wherein said data channel includes switching means for selectively connecting said microprocessor to selected ones of said devices.

4. A computer as defined in claim 3 wherein said switching means comprise a microprocessor controlled switch providing individual interconnection between said microprocessor and any one of said devices or between any two or more of said devices.

5. A computer as defined in claim 3 wherein said switching means comprises a ring circuit connecting each of said interface circuits.

6. A computer as defined in claim 1 wherein said packets include a request-type packet and a reply-type packet, said reply packet being sent by said devices in reply to receipt of said request packet by said device.

7. A computer as defined in claim 6 wherein said request packet includes indicia indicating the size of the reply packet to be returned.

8. A computer as defined in claim 7 wherein each of said data packets includes a header, and said header includes data indicative of the size of the expected reply packet.

9. A computer as defined in claim 7 wherein the size of the reply packet to be sent is dependent on the plurality of prior such transactions, where such said transactions result from similar requests for packets of data.

10. A computer system as defined in claim 1 wherein said interface circuits each include a means whereby the data transmitted from the associated device in a packet remains in storage pending receipt of a reply packet.

11. A computer system as defined in claim 1 wherein said packets include inquiry-type packets reply-type packets, and idle-type packets.

12. A computer system as defined in claim 11 wherein said idle-type packets are periodically sent when one of said devices is not requesting an action from any other said devices or responding to an action from any other said devices.

13. A computer system as defined in claim 1 wherein said interface circuits each include:

a line cache;

said line cache being organized with a variable line size direct mapped cache; and

said cache line is larger than the cache line in the host cache.

* * * * *